

SAMS

畅销全球的  
经典Java教程

“如果您只有一本Java图书，它应该是本书。”

——《个人电脑》杂志，美国最为知名的IT类专业杂志之一

# 21天学通 Java 6

(第5版)

[美] Rogers Cadenhead 著  
Laura Lemay 译  
袁国忠 张劼



人民邮电出版社  
POSTS & TELECOM PRESS

## 内容提要

本书包括 3 周的课程，通过学习，读者将具备使用 Java 开发应用程序、servlet 和 JSP 的知识和技能。第 1 周介绍 Java 语言的基本知识，包括数据类型、变量、表达式、对象、数组、条件语句、循环、类、接口、包、异常、断言和线程等；第 2 周介绍 Java 类库，包括向量、堆栈、映射、散列表和位组等数据结构，以及 Swing 组件、布局管理器和 Java Web Start 等；第 3 周介绍高级主题，包括输入和输出、对象序列化、通过 Internet 进行通信、使用数据库、XML、Web 服务、servlet 和 JSP 等内容。

本书可作为初学者学习 Java 编程技术的教程，也可供已掌握其他语言的程序员学习 Java 时参考。





## 作者简介

Rogers Cadenhead 是 Web 应用程序开发人员兼作者,著有 22 本与 Internet 相关的图书,其中包括《Sams Teach Yourself Java 2 in 24 Hours》。他还是一个网站发布人,其网站每年的访问人次超过 2 400 万。

Laura Lemay 是一名技术文档写作人。在硅谷为各种公司编写 6 年的软件文档后,她发现著书有趣得多。在业余时间里,Laura Lemay 喜欢收集计算机、E-mail 地址、有趣的发色和摩托车模型。她还编写了另外两本图书:《Sams Teach Yourself Web Publishing with HTML》和《Sams Teach Yourself Perl in 21 Days》。



## 致谢

像本书这样涉及范围广泛的书能够得以出版，有赖于很多人的辛勤劳动与奉献，这些人大多是位于印第安纳波利斯的 Sams 出版社的工作人员，他们是 Karen Annett、Adam DeFields、Mandie Frank、Songlin Qiu、Mark Taber 和 Sams 前编辑 Scott Meyers，非常感谢他们；还要感谢 Studio B 的经纪人 Laura Lewin；最重要的是，要感谢我的妻子和儿子们。

还要感谢那些指出本书以前版本的内容和排版错误以及提出改进意见的读者们，他们是 Dave Barton、Patrick Benson、Ian Burton、Lawrence Chang、Jim DeVries、Ryan Esposto、Kim Farr、Sam Fitzpatrick、Bruce Franz、Owen Gailar、Rich Getz、Bob Griesemer、Jenny Guriel、Brenda Henry-Sewell、Ben Hensley、Jon Hereng、Drew Huber、John R Jackson、Bleu Jaegel、Natalie Kehr、Mark Lehner、Stephen Loscialpo、Brad Kaenel、Chris McGuire、Paul Niedenzu、E.J. O'Brien、Chip Pursell、Pranay Rajgarhia、Peter Riedlberger、Darrell Roberts、Luke Shulenburg、Mike Tomsic、John Walker、Joseph Walsh、Mark Weiss、P.C. Whidden、Chen Yan、Kyu Hwang Yeon 和 J-F. Zurcher。

——Rogers Cadenhead

感谢 Sun 公司的 Java 开发小组为开发 Java 语言和浏览器插件付出的辛苦劳动；尤其要感谢 Jim Graham 及时地向我演示 Java 和 HotJava，这埋下了我编写本书的种子；感谢购买我以前的作品并喜欢它们的每位读者，还有购买本书的读者。

——Laura Lemay

## 献词

献给我的儿子 Max、Eli 和 Sam。不管你们在妈妈面前如何评论我的厨艺，我还是为有你们这样的儿子感到非常骄傲。

——Rogers

献给 Eric，为你给予的精神支持和提出的愚蠢问题，还有在我痛苦时泡一大壶咖啡安慰我。

——LL

# 前 言

有些技术革命出其不意地吸引了全世界的眼球。万维网、Linux 操作系统和社会网络的异军突起颠覆了传统的思维模式。

而 Java 语言的巨大成功却在人们的意料之中。自从 Java 语言于 10 年前面世以来,人们就对它充满殷切的期望。当 Sun 公司以融入到 Web 浏览器中的方式推出 Java 时,公众以无比的热情欢迎这种新语言的到来。

Sun 公司创始人 Bill Joy 在介绍这种新语言时,毫不掩饰其孤注一掷的心态:“15 年来,我们一直力图开发出一种更佳的编程语言和环境,用于创建更简单、更可靠的软件,而 Java 就是这种努力的最终结晶。”

在过去的多年中,Java 始终没有辜负媒体的大肆宣传。Java 之于软件开发犹如咖啡之于饮料。咖啡伴随程序员们度过了无数个不眠之夜,而 Java 语言让程序员们完成软件的开发后便可高枕无忧。

最初,Java 是使用在 Web 浏览器中运行的程序来提升网站吸引力的技术;而今天,一些大型网站的服务器仍使用它来驱动关系型数据库支持的动态 Web 应用程序。

每个新的 Java 版本都增强了其作为通用编程语言的功能,使其应用领域超越了 Web 浏览器的范畴。当前,Java 的应用领域涉及桌面应用程序、Internet 服务器、中间件、个人数字助理、嵌入式设备以及众多其他的环境。

现在,Java 语言的第 7 个主要版本——Java 6 完全能够同诸如 C++、Perl、Python、Ruby 和 Visual Basic 等通用开发语言媲美。

您可能熟悉诸如 Eclipse、Borland JBuilder 和 NetBeans 集成开发环境等 Java 编程工具。它们可用于开发 Java 程序,同时您也可以使用 Sun 公司提供的 Java 2 SDK。Java 2 SDK 是一组用于编写、编译和测试 Java 程序的命令行工具,可从网站 <http://java.sun.com> 免费下载。

本书全面地介绍了如何使用最新的 Java 版本和最佳的技术来开发 Java 软件。

阅读本书后,您将知道 Java 语言为何能成为过去 10 年中使用最广泛的编程语言。

## 组织结构

本书介绍 Java 语言以及如何使用它创建可运行在任何计算环境中的应用程序和运行在 Web 服务器中的 servlet。阅读本书后,读者将对 Java 语言和 Java 类库有深入的了解,并能够开发用于完成诸如 Web 服务、数据库连接、处理 XML 和客户/服务器编程等任务的程序。

您将通过实践来学习,在每章中,您都将创建多个程序,这些程序演示了所介绍的主题。本书所有程序的源代码都可在人民邮电出版社网站 ([www.ptpress.com.cn](http://www.ptpress.com.cn)) 下载。

本书包括 3 周课程,分 21 章对 Java 语言及其类库进行了介绍,每周都阐述了开发 Java 小程序和应用程序的一个重要方面。

**第 1 周将介绍 Java 语言本身。**

- 第 1 章介绍基本知识:Java 是什么、为何要学习它以及如何使用面向对象编程技术来创建 Java 程序。

- 第 2 章详细介绍基本的 Java 元素：数据类型、变量和表达式。
- 第 3 章详细阐述了如何在 Java 中处理对象：如何创建对象、如何访问其变量和调用其方法以及如何比较对象。
- 第 4 章将更深入地介绍 Java，包括数组、条件表达式和循环等。
- 第 5 章详细地探讨了如何创建类——所有 Java 程序的基石。
- 第 6 章深入介绍了接口和包，它们对于将类分组以及组织类层次结构很有帮助。
- 第 7 章介绍 Java 中 3 项最强大的功能：异常、线程和断言。异常可用于处理错误，线程用于同时运行程序的各个组成部分，而断言可使程序更可靠。

**第 2 周将介绍 Sun 公司提供的最有用的类，您可以在 Java 程序中使用它们。**

- 第 8 章介绍了可替代字符串和数组的数据结构：向量、堆栈、映射、散列表和位组。还介绍了新增的 for 循环，它使得使用这些数据结构更为容易。
- 第 9 章介绍了如何使用 Swing 来创建图形用户界面。Swing 包含大量的类，用于表示界面、图形和用户交互。
- 第 10 章介绍了十几个可用于 Java 程序中的界面组件，其中包括按钮、文本框、滑块、可滚动的文本区域和图标。
- 第 11 章阐述了如何使用布局管理器来美化用户界面。布局管理器是一组决定组件在界面上如何排列的类。
- 第 12 章阐述了事件处理类，以结束对 Swing 的介绍。事件处理类让程序能够响应鼠标单击和其他用户操作。
- 第 13 章介绍如何在诸如小程序窗口等用户界面组件上绘制几何图形和字符。
- 第 14 章演示如何使用 Java Web Start 和 SwingWorker，前者使得只需单击网页中的链接就可安装 Java 程序，而后者是一个通过使用线程改善应用程序性能类。

**第 3 周介绍高级主题。**

- 第 15 章阐述如何使用流来进行输入和输出。流是让您能够访问文件和网络以及进行其他复杂数据处理的类。
- 第 16 章介绍了对象的序列化——一种让对象在程序没有运行时也能存活的方式。您将学习如何将对象保存到存储介质（如硬盘）中，将其读入到程序并将其作为对象使用。
- 第 17 章将更深入地介绍流以编写能够与 Internet 通信的程序，这包括套接字编程、缓冲区、通道和 URL 处理。
- 第 18 章介绍如何使用 JDBC 和 JDBC-ODBC 连接到数据库。读者将学习如何使用 Java 6 新增的开源数据库 Derby 的功能。
- 第 19 章介绍如何使用 XML 对象模型 (XOM) 和开源 Java 类库读写 RSS 文档。RSS feed 是当前使用最广泛的 XML 方言之一，让数百万用户能够跟踪网站更新和其他新 Web 内容。
- 第 20 章探索如何使用 Java 和 Apache XML-PRC 类库编写 Web 服务客户端。
- 第 21 章介绍 Java 编程中两个最热门的领域：servlet 和 JavaServer Page，这些技术用于编写由 Web 服务器运行的 Java 应用程序。

## 针对的读者

本书针对下列 3 类读者介绍 Java 语言。

- 对编程不太熟悉的新手；

- 早期 Java 版本（如 1.5 或 1.4）的用户；
- 经验丰富的其他语言（如 Visual C++、Visual Basic 或 Python）开发人员。

阅读本书后，读者将熟悉 Java 语言的各个方面，并得心应手地使用 Java 来完成宏大的编程工程——无论是 Web 领域还是其他领域。

如果读者没有编程方面的经验——以前没有编写过程序，可能怀疑本书是否适合您。本书通过程序来阐述所有的概念，因此不管读者的经验是否丰富，都能够理解其中介绍的主题。如果读者熟悉变量、循环和运算符，也将从本书受益。本书的读者分以下几类：

- 学习过 BASIC 或 Pascal，对编程有所了解，并听说 Java 易学、功能强大、很酷；
- 有多年使用其他语言的编程经验，常听到人们赞美 Java，因此想看看是否浪得虚名；
- 听说 Java 在 Web 应用程序和 Web 服务编程方面非常棒。

如果读者不了解面向对象编程——Java 采用的编程模式，也不用担心。本书假设读者没有面向对象设计方面的背景，在您学习 Java 的同时，将了解这种开发方法。

如果读者对编程一无所知，阅读本书时可能会有些吃力。Java 很容易上手，读者只要耐心地阅读，并完成所有的示例，就能够掌握 Java 并开始使用它来编写自己的程序。

## 约定

### 注意

提供与当前的讨论相关的信息，以提醒读者注意（有时涉及技巧方面）。

### 提示

提供针对完成某项工作的建议或更简单的办法。

### 警告

指出潜在的问题，帮助读者远离灾难。

输入的文本或显示在屏幕上的文本使用如下字体：

**It will look like this.**

这种字体类似于文本在屏幕上的外观。变量和表达式占位符使用等宽斜体。

每章最后是与该章主题相关的最常见的问题和作者的回答。另外还有小测验和两个练习，可以帮助读者测试对该章内容的掌握程度。



# 目 录

## 第1周课程 Java 语言

第1章 Java 基础	2	2.4.1 数字字面量	22
1.1 Java 语言	2	2.4.2 布尔字面量	23
1.1.1 Java 的历史	2	2.4.3 字符字面量	23
1.1.2 Java 概述	3	2.4.4 字符串字面量	24
1.1.3 选择一种开发工具	3	2.5 表达式和运算符	24
1.2 面向对象编程	4	2.5.1 算术运算符	25
1.3 对象和类	5	2.5.2 再谈赋值	26
1.4 属性和行为	6	2.5.3 递增和递减运算符	26
1.4.1 对象的类属性	6	2.5.4 比较运算符	27
1.4.2 对象的类行为	7	2.5.5 逻辑运算符	28
1.4.3 创建类	7	2.5.6 运算符优先级	28
1.4.4 运行程序	9	2.6 字符串运算	29
1.5 组织类和类行为	10	2.7 总结	30
1.5.1 继承	10	2.8 问与答	31
1.5.2 创建类层次结构	11	2.9 小测验	31
1.5.3 使用继承	13	2.9.1 问题	31
1.5.4 单继承和多重继承	14	2.9.2 认证练习	32
1.5.5 接口	14	2.10 练习	32
1.5.6 包	14	第3章 对象	33
1.6 总结	15	3.1 创建新对象	33
1.7 问与答	15	3.1.1 使用 new	33
1.8 小测验	15	3.1.2 new 的功能	35
1.8.1 问题	15	3.1.3 内存管理	35
1.8.2 认证练习	16	3.2 访问和设置类变量和实例变量	35
1.9 练习	16	3.2.1 获取值	35
第2章 Java 编程基础	17	3.2.2 修改值	36
2.1 语句和表达式	17	3.2.3 类变量	36
2.2 变量和数据类型	18	3.3 调用方法	37
2.2.1 创建变量	18	3.3.1 嵌套方法调用	38
2.2.2 给变量命名	19	3.3.2 类方法	39
2.2.3 变量类型	19	3.4 对象的引用	39
2.2.4 给变量赋值	20	3.5 对象和基本数据类型的转换和强制类型转换	40
2.2.5 常量	21	3.5.1 强制转换基本类型	41
2.3 注释	22	3.5.2 强制转换对象	42
2.4 字面量	22	3.5.3 基本类型和对象之间的转换	42

3.6 比较对象值和类	43	5.5 Java 应用程序和命令行参数	70
3.6.1 比较对象	44	5.5.1 将参数传递给 Java 应用程序	70
3.6.2 判断对象所属的类	45	5.5.2 在 Java 程序中处理参数	71
3.7 总结	45	5.6 创建名称相同但参数不同的方法	71
3.8 问与答	45	5.7 构造方法	74
3.9 小测验	46	5.7.1 基本的构造方法	74
3.9.1 问题	46	5.7.2 调用另一个构造方法	74
3.9.2 认证练习	46	5.7.3 重载构造方法	75
3.10 练习	47	5.8 覆盖方法	76
第 4 章 数组、逻辑和循环	48	5.8.1 创建覆盖现有方法的方法	76
4.1 数组	48	5.8.2 调用原来的方法	77
4.1.1 声明数组变量	48	5.8.3 覆盖构造函数	77
4.1.2 创建数组对象	49	5.9 结束方法	78
4.1.3 访问数组元素	50	5.10 总结	79
4.1.4 修改数组元素	50	5.11 问与答	79
4.1.5 多维数组	51	5.12 小测验	80
4.2 块语句	52	5.12.1 问题	80
4.3 if 条件语句	52	5.12.2 认证练习	80
4.4 switch 条件语句	53	5.13 练习	81
4.5 for 循环	56	第 6 章 包、接口和其他类特性	82
4.6 while 和 do 循环	58	6.1 限定符	82
4.6.1 while 循环	58	6.2 静态变量和方法	86
4.6.2 do...while 循环	59	6.3 Final 类、方法和变量	87
4.7 跳出循环	60	6.3.1 变量	87
4.7.1 标号	60	6.3.2 方法	88
4.7.2 条件运算符	61	6.3.3 类	88
4.8 总结	61	6.4 抽象类和方法	88
4.9 问与答	61	6.5 包	89
4.10 小测验	62	6.6 使用包	89
4.10.1 问题	62	6.6.1 完整的包名和类名	89
4.10.2 认证练习	62	6.6.2 import 声明	90
4.11 练习	63	6.6.3 类名冲突	91
第 5 章 创建类和方法	64	6.6.4 Classpath 和类的位置	91
5.1 定义类	64	6.7 创建自己的包	91
5.2 创建实例变量和类变量	64	6.7.1 选择包名	91
5.2.1 定义实例变量	64	6.7.2 创建文件夹结构	92
5.2.2 类变量	65	6.7.3 将类加入到包中	92
5.3 创建方法	65	6.7.4 包和类访问控制	92
5.3.1 定义方法	65	6.8 接口	93
5.3.2 关键字 this	66	6.8.1 单继承存在的问题	93
5.3.3 变量作用域和方法定义	67	6.8.2 接口和类	93
5.3.4 将参数传递给方法	68	6.8.3 实现和使用接口	93
5.3.5 类方法	68	6.8.4 实现多个接口	94
5.4 创建 Java 应用程序	69	6.8.5 接口的其他用途	94
		6.9 创建和扩展接口	94



6.9.1 新接口	94	7.3.3 传递异常	110
6.9.2 接口中的方法	95	7.3.4 throws 和继承	111
6.9.3 扩展接口	95	7.4 创建并引发自己的异常	111
6.9.4 创建网上商店	96	7.4.1 引发异常	111
6.10 内部类	100	7.4.2 创建自己的异常	112
6.11 总结	101	7.4.3 组合使用 throws、try 和 throw	112
6.12 问与答	101	7.5 何时使用和不使用异常	113
6.13 小测验	101	7.5.1 什么时候使用异常	113
6.13.1 问题	102	7.5.2 什么时候不使用异常	113
6.13.2 认证练习	102	7.5.3 糟糕的异常使用方式	113
6.14 练习	103	7.6 断言	114
第7章 异常、断言和线程	104	7.7 线程	115
7.1 异常	104	7.7.1 编写线程化程序	116
7.1.1 异常类	105	7.7.2 线程化应用程序	117
7.2 管理异常	106	7.7.3 终止线程	119
7.2.1 异常一致性检测	106	7.8 总结	120
7.2.2 保护代码和捕获异常	106	7.9 问与答	120
7.2.3 finally 子句	108	7.10 小测验	121
7.3 声明可能引发异常的方法	109	7.10.1 问题	121
7.3.1 throws 子句	109	7.10.2 认证练习	122
7.3.2 应引发哪些异常	110	7.11 练习	122

## 第2周课程 Java 类库

第8章 数据结构	124	9.1.2 开发框架	142
8.1 超越数组	124	9.1.3 显示启动画面	143
8.2 Java 数据结构	124	9.1.4 创建组件	143
8.2.1 Iterator	125	9.1.5 将组件加入到容器中	144
8.2.2 位组	126	9.2 使用组件	145
8.2.3 Vector	128	9.2.1 图标	145
8.2.4 遍历数据结构	129	9.2.2 标签	147
8.2.5 堆栈	131	9.2.3 文本框	147
8.2.6 Map	132	9.2.4 文本区域	148
8.2.7 散列表	133	9.2.5 可滚动窗格	149
8.3 泛型	136	9.2.6 复选框和单选按钮	149
8.4 总结	137	9.2.7 组合框	151
8.5 问与答	138	9.2.8 列表	152
8.6 小测验	138	9.3 总结	153
8.6.1 问题	138	9.4 问与答	153
8.6.2 认证练习	139	9.5 小测验	154
8.7 练习	139	9.5.1 问题	154
第9章 使用 Swing	140	9.5.2 认证练习	154
9.1 创建应用程序	140	9.6 练习	155
9.1.1 创建界面	141	第10章 创建 Swing 界面	156

10.1 Swing 的特性 .....	156	12.2.5 鼠标事件 .....	200
10.1.1 设置外观 .....	156	12.2.6 鼠标移动事件 .....	200
10.1.2 标准对话框 .....	156	12.2.7 窗口事件 .....	203
10.1.3 使用对话框 .....	161	12.2.8 使用适配器类 .....	203
10.1.4 滑块 .....	163	12.3 总结 .....	204
10.1.5 滚动窗格 .....	164	12.4 问与答 .....	204
10.1.6 工具栏 .....	165	12.5 小测验 .....	205
10.1.7 进度条 .....	167	12.5.1 问题 .....	205
10.1.8 菜单 .....	168	12.5.2 认证练习 .....	205
10.1.9 选项卡窗格 .....	170	12.6 练习 .....	206
10.2 总结 .....	171	第 13 章 使用颜色、字体和图形 .....	207
10.3 问与答 .....	171	13.1 Graphics2D 类 .....	207
10.4 小测验 .....	172	13.1.1 图形坐标系 .....	208
10.4.1 问题 .....	172	13.1.2 绘制文本 .....	208
10.4.2 认证练习 .....	172	13.1.3 通过反走样改善字体和图形的 质量 .....	210
10.5 练习 .....	173	13.1.4 获取字体的信息 .....	210
第 11 章 在用户界面上排列组件 .....	174	13.2 颜色 .....	211
11.1 基本的界面布局 .....	174	13.2.1 使用 Color 对象 .....	212
11.1.1 布置界面 .....	174	13.2.2 检测和设置当前颜色 .....	212
11.1.2 顺序布局 .....	175	13.3 绘制直线和多边形 .....	213
11.1.3 方框布局 .....	176	13.3.1 用户和设备坐标空间 .....	213
11.1.4 网格布局 .....	177	13.3.2 指定渲染属性 .....	214
11.1.5 边框布局 .....	178	13.3.3 创建要绘制的对象 .....	215
11.2 使用多个布局管理器 .....	179	13.3.4 绘制对象 .....	217
11.3 卡片布局 .....	180	13.4 总结 .....	219
11.4 网格袋布局 .....	184	13.5 问与答 .....	220
11.4.1 设计网格 .....	186	13.6 小测验 .....	220
11.4.2 创建网格 .....	187	13.6.1 问题 .....	220
11.4.3 单元格 padding 和 insets .....	189	13.6.2 认证练习 .....	221
11.5 总结 .....	190	13.7 练习 .....	221
11.6 问与答 .....	190	第 14 章 开发 Swing 应用程序 .....	222
11.7 小测验 .....	191	14.1 Java Web Start .....	222
11.7.1 问题 .....	191	14.2 使用 Java Web Start .....	224
11.7.2 认证练习 .....	191	14.2.1 创建 JNLP 文件 .....	225
11.8 练习 .....	192	14.2.2 在服务器上支持 Web Start .....	228
第 12 章 响应用户输入 .....	193	14.2.3 其他 JNLP 元素 .....	228
12.1 事件监听器 .....	193	14.3 使用 SwingWorker 改善性能 .....	229
12.1.1 设置组件 .....	193	14.4 总结 .....	232
12.1.2 事件处理方法 .....	194	14.5 问与答 .....	232
12.2 使用方法 .....	196	14.6 小测验 .....	233
12.2.1 行为事件 .....	196	14.6.1 问题 .....	233
12.2.2 焦点事件 .....	196	14.6.2 认证练习 .....	233
12.2.3 选项事件 .....	198	14.7 练习 .....	234
12.2.4 键盘事件 .....	199		

## 第3周课程 Java 编程

第15章 输入和输出 .....	236	17.2.1 缓冲区 .....	273
15.1 流 .....	236	17.2.2 字符集 .....	275
15.1.1 使用流 .....	236	17.2.3 通道 .....	275
15.1.2 过滤流 .....	237	17.2.4 网络通道 .....	277
15.1.3 处理异常 .....	237	17.3 总结 .....	281
15.2 字节流 .....	237	17.4 问与答 .....	281
15.3 过滤流 .....	240	17.5 小测验 .....	282
15.4 字符流 .....	246	17.5.1 问题 .....	282
15.4.1 读取文本文件 .....	246	17.5.2 认证练习 .....	282
15.4.2 写文本文件 .....	247	17.6 练习 .....	283
15.5 文件和文件名过滤器 .....	248	第18章 使用 JDBC 访问数据库 .....	284
15.6 总结 .....	250	18.1 JDBC .....	284
15.7 问与答 .....	250	18.2 JDBC-ODBC 桥 .....	285
15.8 小测验 .....	251	18.2.1 连接到 ODBC 数据源 .....	286
15.8.1 问题 .....	251	18.2.2 JDBC 驱动程序 .....	294
15.8.2 认证练习 .....	251	18.3 总结 .....	297
15.9 练习 .....	252	18.4 问与答 .....	297
第16章 序列化和查看对象 .....	253	18.5 小测验 .....	298
16.1 对象序列化 .....	253	18.5.1 问题 .....	298
16.1.1 对象输出流 .....	254	18.5.2 认证练习 .....	298
16.1.2 对象输入流 .....	256	18.6 练习 .....	299
16.1.3 暂态变量 .....	258	第19章 读写 RSS Feed .....	300
16.1.4 检查对象的序列化字段 .....	258	19.1 使用 XML .....	300
16.2 使用反射来检查类和方法 .....	258	19.2 设计 XML 语言 .....	302
16.2.1 检查和创建类 .....	259	19.3 使用 Java 处理 XML .....	303
16.2.2 处理类的各个部分 .....	260	19.4 使用 XOM 处理 XML .....	303
16.2.3 检查类 .....	261	19.4.1 创建 XML 文档 .....	304
16.3 总结 .....	262	19.4.2 修改 MXL 文档 .....	306
16.4 问与答 .....	262	19.4.3 格式化 XML 文档 .....	308
16.5 小测验 .....	263	19.4.4 评估 XOM .....	310
16.5.1 问题 .....	263	19.5 总结 .....	311
16.5.2 认证练习 .....	263	19.6 问与答 .....	312
16.6 练习 .....	264	19.7 小测验 .....	312
第17章 通过 Internet 进行通信 .....	265	19.7.1 问题 .....	312
17.1 JAVA 的联网技术 .....	265	19.7.2 认证练习 .....	312
17.1.1 打开跨越网络的流 .....	265	19.8 练习 .....	313
17.1.2 套接字 .....	268	第20章 XML Web 服务 .....	314
17.1.3 Socket 服务器 .....	270	20.1 XML-RPC 简介 .....	314
17.1.4 设计服务器应用程序 .....	271	20.2 使用 XML-RPC 进行通信 .....	315
17.1.5 测试服务器 .....	272	20.2.1 发送请求 .....	315
17.2 java.nio 包 .....	273		

20.2.2 响应请求	316	21.2 开发 Servlet	328
20.3 选择 XML-RPC 实现	317	21.2.1 使用 cookie	331
20.4 使用 XML-RPC Web 服务	318	21.2.2 使用会话	334
20.5 创建 XML-RPC Web 服务	320	21.3 JSP	336
20.6 总结	323	21.3.1 编写 JSP	337
20.7 问与答	323	21.3.2 创建 Web 应用程序	342
20.8 小测验	324	21.4 JSP 标准标记库	346
20.8.1 问题	324	21.5 总结	350
20.8.2 认证练习	324	21.6 问与答	350
20.9 练习	325	21.7 小测验	351
第 21 章 编写 Java Servlet 和 Java Server		21.7.1 问题	351
Page	326	21.7.2 认证练习	351
21.1 使用 Web Servlet	326	21.8 练习	352

## 附 录

附录 A 使用 Java 开发包	354	附录 B 使用 Java 开发包编程	369
A.1 选择 Java 开发工具	354	B.1 JDK 概览	369
A.2 配置 JDK	356	B.2 Java 解释器	370
A.2.1 使用命令行界面	356	B.3 编译器 Javac	371
A.2.2 切换文件夹	357	B.4 浏览器 appletviewer	372
A.2.3 在 MS-DOS 中创建文件夹	358	B.5 文档工具 Java doc	374
A.2.4 在 MS-DOS 中运行程序	359	B.6 Java 文件存档工具 Jar	376
A.2.5 修复配置错误	360	B.7 调试器 Jdb	377
A.3 使用文本编辑器	362	B.7.1 调试应用程序	378
A.4 创建程序	363	B.7.2 调试小程序	379
A.4.1 在 Windows 中编译和运行程序	364	B.7.3 高级调试命令	379
A.4.2 设置 CLASSPATH 变量	365	B.8 使用系统属性	380





## 第1周课程

### Java 语言

第1章 Java 基础

第2章 Java 编程基础

第3章 对象

第4章 数组、逻辑和循环

第5章 创建类和方法

第6章 包、接口和其他类特性

第7章 异常、断言和线程

# 第 1 章

## Java 基础

Java 试图解决众多领域的问题，实际上也确实在这方面取得了极大的成功。它让程序员能够开发应用程序服务器和手机程序、进行科学编程、编写软件以及星际导航等。

——Java 语言之父 James Gosling 在接受 SearchWebServices.com 采访时如是说

当 Sun 公司于 1995 年首次发布 Java 编程语言时，Java 是一个用于万维网的颇具创意的玩具，但有很大的发展潜力。

“潜力”是一个有时限的恭维之词。潜力迟早需要变成现实，否则将被“衰弱”、“浪费”、“失望”等取代。

通过阅读本书，读者在提高自身技能的同时，将能够对 Java 语言是否像它多年来被宣传的那样做出客观的判断。

读者还将成为极具潜力的 Java 程序员。

### 1.1 Java 语言

现在的 Java 是第 7 个主要版本，它没有辜负当时人们对它的期望。在诸如 NASA、IBM、Kaiser Permanente 和 Apache Project 等企业和组织中，有超过 350 万的程序员学习了该语言并正在使用它。遍布世界各地的众多大学的计算机科学系将其列为标准教学课程。Java 最初用于在网页中创建简单程序，而现在已被用于众多的领域，其中包括：

- Web 服务器；
- 关系型数据库；
- 轨道望远镜；
- 个人数字助理；
- 手机。

对于那些希望其网站栩栩如生以及创建 Web 应用程序的 Web 开发人员来说，Java 仍很有帮助，但其应用领域已远远超出了 Web，成为一种流行的通用编程语言。

#### 1.1.1 Java 的历史

现在，有关 Java 语言的故事已是家喻户晓。20 世纪 90 年代中期，Sun 公司的 James Gosling 和其他开发人员正致力于开发一个交互式 TV 项目，Gosling 对正在使用的 C++感到失望。C++是一种面向对象编程语言，它是由 AT&T 的贝尔实验室的 Bjarne Stroustrup 于 10 年前在 C 语言的基础上开发的。

Gosling 把自己关在办公室，创建了一种适合其项目的语言，该语言解决了 C++中一些令其失望的问题。

Sun 公司的交互式 TV 项目以失败告终,但出乎人们意料的是,在此期间开发出来的新语言却适用于此时逐渐流行的一种新媒体——万维网。

1995 年秋天, Sun 公司首次发布了 Java。虽然与 C++ (以及当今的 Java) 相比, 该语言的大多数特性太初级, 但被称作小程序 (applets) 的 Java 程序可作为网页的一部分运行在 Netscape Navigator 浏览器中。

这种功能——第一种用于 Web 的交互式编程技术——给这种新语言提供了极大的舆论攻势, 在短短的 6 个月内便吸引了数以十万计的开发人员。

在人们对 Java Web 编程技术的好奇过后, 该语言的整体优势逐渐明朗, 程序员们仍在继续使用它。有些调查表明, 当前职业 Java 程序员人数超过了 C++ 程序员。

### 1.1.2 Java 概述

Java 是一种面向对象的、独立于平台的安全语言, 它比 C++ 更容易学习, 且比 C 和 C++ 更能避免被误用。

面向对象编程 (OOP) 是一种软件开发方法, 它将程序视为一组协同工作的对象。对象是使用被称作类的模板创建的, 它们由数据和使用数据所需的语句组成。Java 是完全面向对象的, 在本章后面, 当您创建第一个类并使用它来创建对象时将明白这一点。

独立于平台指的是无需修改程序便能够运行在不同的计算环境中。Java 程序被编译成一种名为字节码的格式, 字节码可被任何带有 Java 解释器的操作系统、软件或设备运行。您可以在 Windows Vista 机器上创建 Java 程序, 然后在 Linux Web 服务器、使用 OS X 的 Apple Mac 和 Palm 个人数字助理上运行它。只要平台安装了 Java 解释器, 便可以运行字节码。

虽然比其他语言更容易学习是程序员们争论的焦点之一, 但 Java 主要在以下方面比 C++ 更容易:

- Java 自动负责内存的分配和释放, 将程序员从这种烦琐、复杂的工作中解放出来;
- Java 没有指针——指针是一种功能强大的特性, 主要供经验丰富的程序员使用, 不过他们也容易误用该特性;
- Java 只具备面向对象编程中的单继承。

Java 之所以安全的两个关键因素是没有指针且能自动管理内存。

### 1.1.3 选择一种开发工具

介绍 Java 后, 接下来应用其中的一些概念, 创建您的第一个 Java 程序。

如果读者从头到尾阅读本书后, 将对 Java 的功能有深入的了解, 其中包括图形、文件输入和输出、Web 应用程序开发、可扩展标记语言 (XML) 处理和数据库连接。您将能编写运行在网页、个人计算机和 Web 服务器以及其他计算环境中的 Java 程序。

开始编写程序之前, 您必须在计算机上安装用于编辑、编译和运行 Java 程序 (这些程序使用的是最新的 Java 版本 Java 6) 的软件。

支持 Java 6 的 Java 集成开发环境有很多, 如 Borland JBuilder、IntelliJ IDEA 和 Eclipse。

这些集成开发环境都被 Java 开发人员所极力推荐, 但如果您在学习 Java 语言的同时学习使用这些工具, 将是一项非常艰巨的任务。大多数集成开发环境主要针对的是需要提高效率的、经验丰富的程序员, 而不是刚开始学习一门新语言的新手。

因此, 除非您阅读本书之前已经熟练使用某种开发工具, 否则应使用最简单的 Java 开发工具——Java 开发包, 该工具包可从 Sun 公司的 Java 网站 (<http://java.sun.com>) 免费下载。

每当 Sun 公司发布新的 Java 版本时, 都会在网上提供一个支持该版本的免费开发包。最新的版本



为 Java Development Kit Version 6。

出于简化的目的，本书通常将该语言及其开发包简称为 Java 和 JDK。在其他地方，该软件包可能被称作 Java 开发包 6。

如果您打算使用 JDK 来创建本书的程序，请参阅附录 A 以了解有关如何使用该软件的基本知识。该附录介绍了如何下载并安装该开发包以及如何使用它来创建 Java 程序。

在计算机上安装了支持 Java 6 的 Java 开发工具后，便可以开始学习使用该语言了。

## 1.2 面向对象编程

对新的 Java 程序员来说，最大的挑战在于学习该语言的同时学习面向对象编程。

如果您不熟悉这种编程方式，这听起来可能有些令人沮丧，虽然如此，但您可以把它当作是一种买一赠一的回馈。您将通过学习 Java 来掌握面向对象编程技术，否则，您将无法使用这种语言。

面向对象编程是一种创建计算机程序的方法，它模仿了现实世界中物体被组合在一起的方式。

使用这种开发风格，可以创建出更可靠、更容易理解、可重用性更高的程序。

为此，必须首先研究 Java 是如何实现面向对象编程原理的。在本书的第 1 周课程中，将介绍如下主题：

- 将程序组织为叫做类的元素；
- 了解如何用这些类来创建对象；
- 通过类结构的两个方面（行为和属性）来定义类；
- 通过继承将类彼此关联起来；
- 通过包和接口将类关联起来。

如果您熟悉面向对象编程，本章的很多内容将起到温故知新的作用。即使跳过那些介绍性内容，也应创建示例程序，以积累一些开发、编译和运行 Java 程序的经验。

概念化计算机程序的方式很多，其中之一是将程序视为一系列依次执行的指令，这通常被称为过程化编程。很多程序员开始学习的都是过程化语言，如 BASIC。

过程化语言模仿了计算机执行指令的方式，因此程序与计算机执行任务的方式一致。过程化程序员首先必须学习如何将问题分解为一系列的简单步骤。

面向对象语言从另一个角度来看待计算机程序，它将重点放在您要求计算机完成的任务，而不是计算机完成任务的方式。

在面向对象编程中，计算机程序被视为一组相互协同、共同完成任务的对象。每个对象都是程序的独立部分，它以特定的、高度可控制的方式与其他部分进行交互。

在现实生活中，一个面向对象设计的例子是立体声音响系统。大多数系统都是通过将一组不同的对象组合在一起而构建起来的，这些对象通常被称为组件，如：

- 音箱用于播放中频和高频的声音；
- 低音喇叭用于播放低频声音；
- 调谐器用于接收无线电广播信号；
- CD 播放器用于读取光盘中的音频数据。

这些组件能够通过标准的输入/输出端子进行彼此交互。即使您买的音箱、低音喇叭、调谐器和 CD 播放器不是同一个厂家的，只要它们有标准端子，就可以将它们组合成一个立体声音响系统。

面向对象编程的工作原理与此相同：您可以用标准方式将新创建的对象和已有的对象组合成一个程序，其中每个对象在整个程序中扮演着特定的角色。

对象 (object) 是计算机程序中的一个自包含元素，它包含一组相关的特性，能完成特定的任务。

## 1.3 对象和类

面向对象编程是基于现实世界的情况进行建模的，对象由多种更小的对象构成。

然而，合并对象仅仅是面向对象编程的一个方面，其另一个重要特征是使用类。

类（class）用来创建对象的模板。使用同一个类创建出来的每个对象都具有就算不是完全相同也相似的特性。

类包含一组特定对象的所有特性。使用面向对象语言编写程序时，并不定义各个对象，而是定义用于创建这些对象的类。

例如，您可能创建 Modem 类，它描述了所有计算机调制解调器的特征，其中一些常见的特征有：

- 连接到计算的串行端口；
- 发送和接收信息；
- 拨叫电话号码。

Modem 类是调制解调器的抽象概念模型。要在程序中有能够实际操纵的具体东西，必须用 Modem 类来创建 Modem 对象。使用类创建对象的过程叫做实例化（instantiation），这就是对象也被称作实例的原因所在。

Modem 类可以用来创建很多不同的 Modem 对象，其中每个对象都可以有不同的特征，如：

- 有些调整解调器是内置的，而有些是外置的；
- 有些使用 COM1 端口，有些使用 COM2 端口；
- 有些有差错控制功能，而其他的没有。

即使有这么多的不同，两个 Modem 对象仍有足够多的共性，使其被视为相关的对象。图 1.1 显示了 Modem 类及使用该模板创建的几个对象。

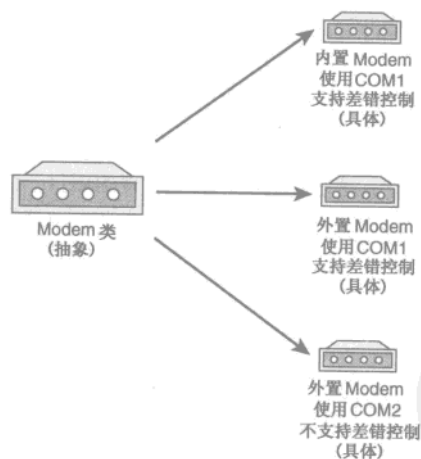


图 1.1 Modem 类和几个 Modem 对象

这里还有一个例子：使用 Java 可以创建一个类来表示所有的命令按钮——可单击的框，出现在窗口、对话框和程序图形用户界面的其他部分。

开发 CommandButton 类时，可以定义如下特征：

- 显示在按钮上的文本；
- 按钮的大小；
- 按钮的外观，如是否有三维阴影效果。

CommandButton 类还可以定义按钮的行为，如：

- 按钮需要单击还是双击；
- 是否完全忽略鼠标单击；
- 被单击后执行什么操作。

定义 `CommandButton` 类后，就可以创建按钮实例了，即 `CommandButton` 对象。这些对象都具有类中定义的可单击按钮的基本特征，但根据需要，每个对象都可以有不同的外观和行为。

通过创建 `CommandButton` 类，可避免为程序中使用每个命令按钮重写这些代码。此外，如果需要的话，还可以重用该 `CommandButton` 类来创建不同类型的按钮，无论是在当前程序中还是在其他程序中。

**注意**

Java 的标准类 `javax.swing.JButton` 包含了这个虚构的 `CommandButton` 类的所有功能，同时还有其他功能。您将在第 9 章中使用它。

编写 Java 程序时，您实际上设计和构建了一组类。程序运行时，将根据需要使用这些类来实例化对象，并使用它们。作为 Java 程序员，您的任务是创建一组合适的类，来完成程序要完成的任务。

幸运的是，不必每次从头开始。Java 语言包含了数千个类，它们实现了您所需的大部分基本功能。这些类被称为 Java 类库，它们随诸如 JDK 等开发工具一起被安装。

当您谈论如何使用 Java 语言时，实际谈论的是有关使用该类库和 Java 编译器能够识别的标准关键字和运算符。

类库处理很多任务，如数学函数、文本处理、图形、用户交互以及网络功能等。使用这些类与您自己创建的 Java 类没有什么不同。

对于复杂的 Java 程序，可能需要创建一整套新类，这些类可以用于构建您自己的类库，以便在其他程序中使用。

重用是面向对象编程的基本优点之一。

## 1.4 属性和行为

Java 类包含两种不同的信息：属性和行为。

这两者在 `VolcanoRobot` 中都有，这是今天将作为类实现的项目。该项目使用计算机模拟火山探测工具，它模仿的是 NASA 的“遥控机器人研究”计划中用来在火山裂缝中进行研究探测的 Dante II 机器人。

### 1.4.1 对象的类属性

属性 (attributes) 是对象区别于其他对象的数据，可用于确定属于该类的对象的外观、状态和其他性质。

火山探测工具可能有如下属性。

- 状态：探测、移动、返回。
- 速度：以每小时的英里数计量。
- 温度：以华氏温度计量。

在类中，属性是通过变量定义的，变量是计算机程序中用来存放信息的位置。实例变量是这样的属性，即它们的值随对象而异。

实例变量 (instance variable) 定义了特定对象的属性。对象的类定义了属性的类型，每个实例都存储了自己的属性值。实例变量也叫做对象变量 (object variable)。

每个类属性都有一个相应的变量，可以通过修改该变量的值来修改对象的属性。

例如, `VolcanoRobot` 类定义了一个名为 `speed` 的实例变量。这必须定义成一个实例变量, 因为每个机器人都将根据当前环境以不同的速度运动。可以通过修改机器人的 `speed` 实例变量, 使该机器人更快或更慢地移动。

创建对象时, 可以给实例变量赋值, 并在对象的整个生命期中保持不变; 也可以在程序运行过程中使用该对象时, 给它指定不同的值。

对于其他变量, 让类的所有对象共享同一个值更合理, 这些属性叫做类变量。

类变量 (class variable) 定义了类的属性。该变量用于类本身及其所有的实例, 因此不管使用该类创建了多少个对象, 都只存储该变量的一个值。

在 `VolcanoRobot` 类中, 存储当前时间的变量就是一个类变量。如果使用实例变量来存储当前时间, 则在每个对象中, 该变量的值都可能不同。这样, 如果这些机器人要协同完成某项任务, 就可能出现

问题。

使用类变量可以避免这种问题, 因为类的所有对象共享相同的值。每个 `VolcanoRobot` 对象都有权访问该变量。

### 1.4.2 对象的类行为

行为 (behavior) 指的是对象能够对自身和其他对象执行的操作。行为可以用来修改对象的属性、接收来自其他对象的信息以及向其他对象发送消息让它们执行任务。

火山机器人可能有如下行为:

- 检查当前温度;
- 开始勘测;
- 报告当前位置。

对象类的行为是使用方法来实现的。

方法 (method) 是对象类中用来完成特定任务的一组相关语句。它们用来针对对象本身或其他对象执行特定任务, 相当于其他编程语言中的函数和子程序。

对象间能够通过方法彼此进行通信。类或对象可能调用其他类和对象的方法, 其原因很多, 例如:

- 将变化告知另一个对象;
- 让其他对象对自身进行修改;
- 让其他对象执行某项操作。

例如, 两个火山机器人可以使用方法来报告彼此的位置, 以避免碰撞; 一个机器人可以要求另一个停下来, 以便它能够顺利通过。

正像变量分为实例变量和类变量一样, 方法也分为实例方法和类方法。实例方法 (instance method) 很常见, 因此它们通常被简称为方法, 它们用于处理类的一个对象。如果一种方法对单个对象进行修改, 则它必须是实例方法。类方法 (class method) 适用于类本身。

### 1.4.3 创建类

为说明类、对象、属性和行为, 您将开发一个 `VolcanoRobot` 类、使用这个类创建对象并在程序中使用它们。

#### 注意

该程序的主旨是探索面向对象编程。有关 Java 编程语法的更详细的信息, 请参阅第 2 章。

要创建类, 请运行用来创建 Java 程序的文本编辑器, 并新建一个文件。然后, 输入程序清单 1.1

中的代码，将其保存在用来存放本书程序的文件夹中，并将其命名为 `VolcanoRobot.java`。

程序清单 1.1 `VolcanoRobot.java` 的源代码

```

1: class VolcanoRobot {
2:     String status;
3:     int speed;
4:     float temperature;
5:
6:     void checkTemperature() {
7:         if (temperature > 660) {
8:             status = "returning home";
9:             speed = 5;
10:        }
11:    }
12:
13:    void showAttributes() {
14:        System.out.println("Status: " + status);
15:        System.out.println("Speed: " + speed);
16:        System.out.println("Temperature: " + temperature);
17:    }
18: }
```

第 1 行的 `class` 语句定义了一个名为 `VolcanoRobot` 的类。从第 1 行的花括号到第 18 行的花括号之间的所有东西都属于这个类。

`VolcanoRobot` 类包含 3 个实例变量和 2 个实例方法。

实例变量是在第 2~4 行定义的：

```
String status;
int speed;
float temperature;
```

变量名为 `status`、`speed`、`temperature`，其中每个变量都将用来存储一种不同类型的信息。

- `status`：存储一个 `String` 对象——组字母、数字、标点和其他字符。
- `speed`：存储一个 `int` 对象，即整数值。
- `temperature`：存储一个 `float` 对象，即浮点数值。

`String` 对象是使用 `String` 类创建的，后者位于 Java 类库中，可用于任何 Java 程序中。

**提示** 从该程序可知，类也可以将对象作为实例变量。

`VolcanoRobot` 类的第一个实例方法是在第 6~11 行定义的：

```
void checkTemperature() {
    if (temperature > 660) {
        status = "returning home";
        speed = 5;
    }
}
```

方法的定义方式与类相似，首先是指定方法名称、返回值和其他内容的语句。

`checkTemperature()` 方法位于第 6 和 11 行的花括号之间。可对 `VolcanoRobot` 对象调用该方法，以确定其温度。

该方法检查对象的 `temperature` 实例变量的值是否大于 660。如果是，则修改另外两个实例变量：

- 将 `status` 修改为“returning home”，这表明温度太高，机器人应返回基地；
- 将 `speed` 修改为 5（假设这是机器人的最快速度）。

第二个实例方法——`showAttributes()` 是在第 13~17 行定义的：

```
void showAttributes() {
    System.out.println("Status: " + status);
    System.out.println("Speed: " + speed);
    System.out.println("Temperature: " + temperature);
}
```

这种方法使用 `System.out.println()` 来显示 3 个实例变量的值，同时显示一些文本用于解释每个值的含义。

输入源代码后，将其保存。现在还不需要编译它。

#### 1.4.4 运行程序

如果此时编译 `VolcanoRobot` 类，也无法使用它来模拟探险机器人。您所创建的类定义了 `VolcanoRobot` 对象，但没有使用这种对象。

使用 `VolcanoRobot` 类的方式有两种：

- 创建一个独立的 Java 程序，并在其中使用这个类；
- 在 `VolcanoRobot` 类中添加一个特殊的类方法——`main()`，使之能作为一个应用程序而运行，然后在该方法中使用 `VolcanoRobot` 对象。

这里使用第一种方法。程序清单 1.2 是 Java 类 `VolcanoApplication` 的源代码，它创建一个 `VolcanoRobot` 对象、设置其实例变量并调用其方法。

程序清单 1.2 最后的 `VolcanoApplication.java` 源代码

```
1: class VolcanoApplication {
2:     public static void main(String[] arguments) {
3:         VolcanoRobot dante = new VolcanoRobot();
4:         dante.status = "exploring";
5:         dante.speed = 2;
6:         dante.temperature = 510;
7:
8:         dante.showAttributes();
9:         System.out.println("Increasing speed to 3.");
10:        dante.speed = 3;
11:        dante.showAttributes();
12:        System.out.println("Changing temperature to 670.");
13:        dante.temperature = 670;
14:        dante.showAttributes();
15:        System.out.println("Checking the temperature.");
16:        dante.checkTemperature();
17:        dante.showAttributes();
18:    }
19: }
```

将该程序保存为 `VolcanoApplication.java`，并编译它。

如果您使用的是 JDK，可以这样进行编译：切换到命令行或打开一个命令行窗口，然后切换到 `VolcanoApplication.java` 所在的目录，并在命令行中执行下述命令：

```
javac VolcanoApplication.java
```

Java 编译器创建一个名为 `VolcanoApplication.class` 的文件，其中包含可被 Java 解释器执行的字节码。必要时，Java 编译器也将编译文件 `VolcanoRobot.java`，因为这个应用程序使用了 `VolcanoRobot` 类。

编译该应用程序后，便可运行它。

使用 JDK 时，要运行 `VolcanoApplication` 应用程序，可在命令行中切换到文件 `VolcanoRobot.class` 和 `VolcanoApplication.java` 所在的文件夹，然后执行下述命令：

```
java VolcanoApplication
```

运行 `VolcanoApplication` 时，其输出如下：

```

java VolcanoApplication
Status: exploring
Speed: 2
Temperature: 510.0
Increasing speed to 3.
Status: exploring
Speed: 3
Temperature: 510.0
Changing temperature to 670.
Status: exploring
Speed: 3
Temperature: 670.0
Checking the temperature.
Status: returning home
Speed: 5
Temperature: 670.0

```

根据程序清单 1.2 可知，类方法 `main()` 执行了以下操作。

- 第 2 行：创建并命名 `main()` 方法。所有 `main()` 方法的格式都与此相同，有关这方面的内容将在第 5 章中做更详细的介绍。现在，您需要注意的是关键字 `static`，它表明该方法是一个类方法，供所有 `VolcanoRobot` 对象共享。
- 第 3 行：使用 `VolcanoRobot` 类为模板创建了一个新的 `VolcanoRobot` 对象。该对象被命名为 `dante`。
- 第 4~6 行：给对象 `dante` 的实例变量赋值——`status` 被设置为 `exploring`，`speed` 为 2，`temperature` 为 510。
- 第 8 行：在该行及随后的几行中，调用了 `dante` 对象的 `showAttributes()` 方法。这个方法显示实例变量 `status`、`speed` 和 `temperature` 的当前值。
- 第 9 行：在该行和随后的几行中，使用 `System.out.println()` 语句来显示圆括号内的文本。
- 第 10 行：将 `speed` 实例变量的值设置为 3。
- 第 13 行：将 `temperature` 实例变量的值设置为 670。
- 第 16 行：调用 `dante` 对象的 `checkTemperature()` 方法。该方法检查实例变量 `temperature` 的值是否大于 660。如果是，则将新的值赋给 `status` 和 `speed`。

## 1.5 组织类和类行为

如果不涉及继承、接口和包这 3 个概念，对 Java 的面向对象编程的介绍将是不完整的。它们都是用于组织类和类行为的机制。

### 1.5.1 继承

继承是面向对象编程中最重要的概念之一，它直接影响您如何设计和编写 Java 类。

继承是一种机制，让一个类能够继承另一个类的所有行为和属性。

通过继承，一个类可以拥有已有类的所有功能。因此，只需指明新类与现有类的不同。

通过继承，所有的类（无论是您创建的类还是 Java 类库和其他库中的类）都以严格的层次结构来组织。

继承其他类的类叫做子类，被继承的类叫做超类。

一个类只能有一个超类，但每个类都可以有任意数目的子类。子类继承了其超类的所有属性和行为。

实际上，这意味着如果超类具备您的类所需的行为和属性，则无需重新定义或复制代码，便可获得同样的行为和属性。子类将自动从其超类那儿获得这些东西，而超类又从其超类获得相应的东西，



依此类推。这样便形成了层次结构。子类将拥有层次结构中位于它上面所有类的特性，同时也有自己的特性。

这与您从父母那里继承各种东西（如身高、头发颜色、喜欢花生黄油和香蕉三明治）相同。它们也从其父母那里继承了一些特征，它们的父母又是从它们父母的父母那里继承下来的，这样一直追溯到伊甸园、宇宙大爆炸或大爆炸之前。

图 1.2 显示了类的层次排列方式。

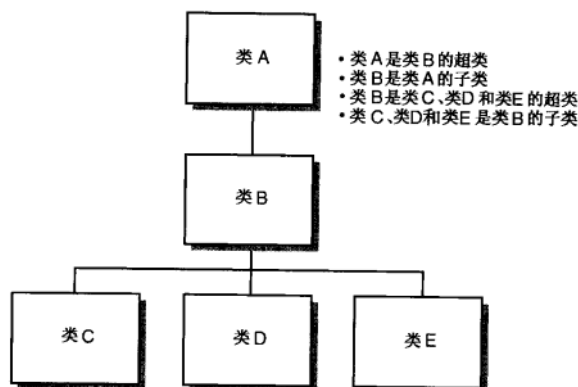


图 1.2 类层次结构

在 Java 类层次结构的顶端是类 `Object`——所有的类都是从这个超类继承而来的。`Object` 是层次结构中最通用的类，Java 类库中的所有类都继承了它定义的行为。

在层次结构中，越往下，类的用途越具体。在层次结构的顶部定义的是抽象概念，越往下，这些概念越具体。

使用 Java 创建新类时，常常希望它具备某个现有类的所有功能，并做一些修改。例如，您可能希望有一个 `CommandButton`，它能够在单击时发出声音。

要经过任何重建工作而得到 `CommandButton` 的所有功能，可以将您的类定义为 `CommandButton` 的子类。这样，您的类将自动继承 `CommandButton` 定义的行为和属性以及 `CommandButton` 的超类定义的行为和属性。您所需要关心的只是新类不同于 `CommandButton` 的内容。子类化 (subclassing) 机制用于定义新类及其与超类之间的差别。

子类化指的是通过继承已有的类来创建一个新类。子类只需指出其属性和行为不同于超类的地方。

如果您的类定义了全新的行为，且不是其他类的子类，则可以直接继承 `Object` 类。这使其能够进入 Java 类层次中。实际上，如果您创建类定义时没有指定超类，Java 认为这个新类将直接继承 `Object`。前面创建的 `VolcanoRobot` 类就是从 `Object` 派生而来的。

### 1.5.2 创建类层次结构

如果您创建了大量的类，则应该让您的类从现有的类层次结构中继承，并构建自身的层次结构。这将有如下优点：

- 可将多个类共有的功能放在一个超类中，这样就可以在更底层的类中重复使用这些功能；
- 对超类的修改将自动反映到其所有的子类、子类的子类等中，而无需修改或重新编译更底层的类，它们将通过继承获得新的信息。

例如，假设创建了一个 Java 类来实现火山勘测机器人的所有特征（这并不需要太多的想象力）。

该 `VolcanoRobot` 类已经完成，它工作正常，一切都很好。现在您要创建一个名为 `MarsRobot` 的

Java 类。

这两种机器人有相似的特征：都是在恶劣的环境下执行研究工作的机器人。您首先的想法也许是打开 `VolcanoRobot.java` 源文件，将其中的大部分内容复制到一个名为 `MarsRobot.java` 的新源文件中。

更好的办法是找出 `MarsRobot` 和 `VolcanoRobot` 的共同功能，并将它们放到一个更通用的类层次结构中。对于只有类 `VolcanoRobot` 和 `MarsRobot` 的情况，这也许是项繁重的工作，但如果您还想加入 `MoonRobot`、`UnderseaRobot` 和 `DesertRobot`，情况将如何呢？将共同的行为放在一个或多个可重用的超类中将极大地减少所需完成的工作量。

要设计一个满足该目标的类层次，应从 `Object` 开始，它是所有 Java 类的祖宗。这些机器人的老祖宗可能名为 `Robot`。一般而言，机器人可以被视为一个自控的探测设备。在 `Robot` 类中，您只定义使其成为自控的、用于探测的设备的行为。

在 `Robot` 下面有两个类：`WalkingRobot` 和 `DrivingRobot`。这两个类之间明显的区别在于一个靠腿移动，另一个靠轮子移动。步行机器人的行为可能包括弯腰捡东西、蹲下、跑动等。驱动式机器人的行为与此不同。图 1.3 显示了您目前已有的类层次结构。

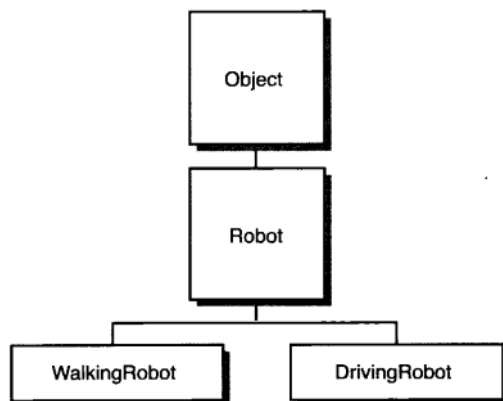


图 1.3 基本的 Robot 层次结构

现在，这个层次可以更具体。从 `WalkingRobot` 类可以派生出多种类：`ScienceRobot`、`GuardRobot`、`SearchRobot` 等。另外，您可以抽取更多的功能，并创建两个作为中间类的 `TwoLegged` 和 `FourLegged`，其中每个类都有不同的行为（见图 1.4）。

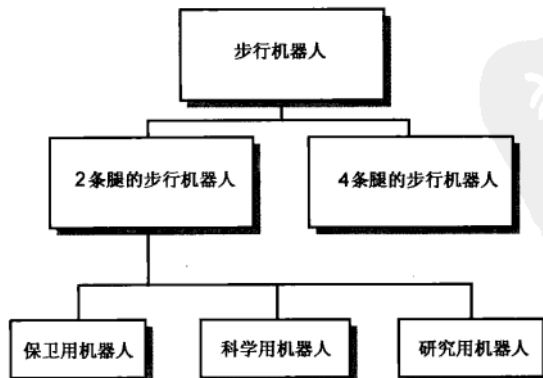


图 1.4 2 条腿和 4 条腿的步行机器人

最后，整个层次结构便完成了，并为 VolcanoRobot 找到了合适的位置。它可以是 ScienceRobot 的子类，ScienceRobot 是 WalkingRobot 的子类，WalkingRobot 是 Robot 的子类，而 Robot 又是 Object 的子类。

诸如 status、temperature 和 speed 等属性应放在什么位置呢？应放在最合适的地方。因为所有的机器人都需要跟踪其所在环境的温度，因此在 Robot 中将 temperature 定义为一个实例变量是合理的。这样所有的子类都将有这个实例变量。请记住，只需在层次结构定义行为或属性一次，它将自动被每个子类继承。

#### 注意

要设计出高效的类层次结构，需要做大量的规划和修订。当您试图将新的属性和行为加入到层次结构中时，很可能发现需要将一些类移到另一个位置，以便减少重复的特征。

### 1.5.3 使用继承

在 Java 中，继承比现实生活中的继承要简单得多。Java 中不需要遗嘱执行人、法官，也不需要法庭。

当您创建一个新的对象时，Java 将记录该对象及其超类的每个变量。这样，所有的类组合成当前对象的模板，每个对象都将包含合适的信息。

方法的工作原理与此相似，新对象可以访问其所属类及其超类中的所有方法，这是在运行期间当方法被使用时动态确定的。如果您调用了特定对象的某个方法，Java 解释器将首先检查该对象所属的类是否有该方法。如果没有，则在其超类中查找，依次类推，直到找到该方法的定义为止。图 1.5 所示对此做了说明。

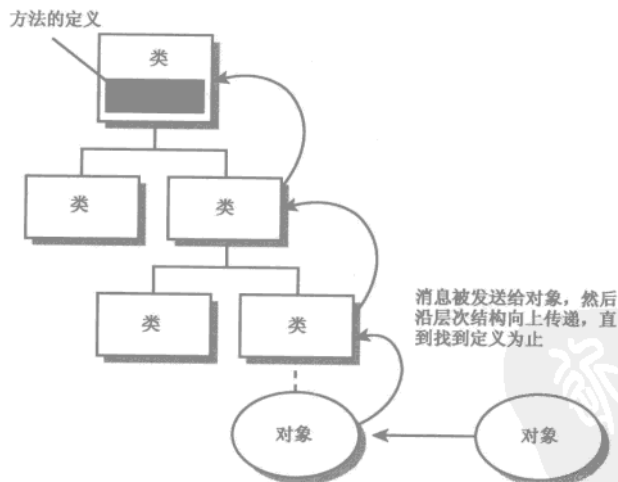


图 1.5 如何在类层次结构中查找方法

如果子类中定义了名称和其他方面都与超类相同的方法，情况将复杂起来。在这种情况下，首先被找到的方法是被使用的方法（从层次结构的底部开始向上查找）。

因此，可以在子类中创建一个方法来防止调用超类中定义的方法。为此，该方法的名称、返回值和参数必须与超类方法相同。这被称为覆盖，如图 1.6 所示。

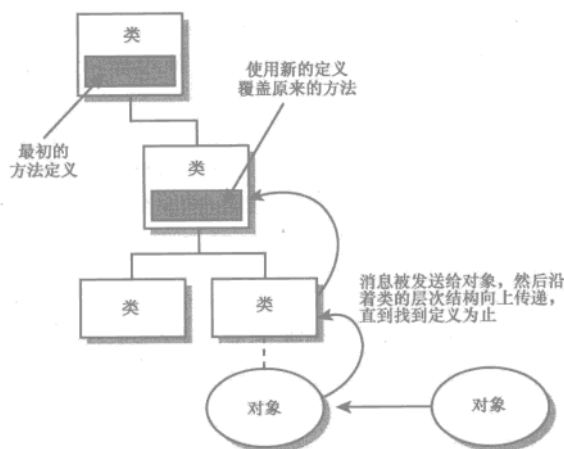


图 1.6 覆盖方法

### 1.5.4 单继承和多重继承

Java 的继承形式称为单继承 (single inheritance)，因为每个 Java 类都只能有一个超类（虽然任何超类都可以有多个子类）。

在其他面向对象编程语言（如 C++）中，类可以有多个超类，子类将继承所有超类的变量和方法。这叫做多重继承 (multiple inheritance)，使用这种方法可以创建包含所有想象得到的行为的类。然而它也使得类的定义和所需的代码相当复杂。Java 只允许单继承，从而简化了继承机制。

### 1.5.5 接口

单继承简化了类之间的关系，并使这些类实现的功能更容易理解和设计。然而，它也有局限性，尤其是当您有一些相似的行为，它们需要在类层次结构中不同的分支间进行复制时。Java 通过使用接口来解决这些共享行为的问题。

接口是一组方法，它指出类除了从超类继承的行为外，还有其他行为。接口中的方法并没有定义行为，这项任务将由实现该接口的类去完成。

例如，Comparable 接口包含一个这样的方法，它对属于同一个类的两个对象进行比较，以判断在排序链表中，哪个应在前面。任何实现该接口的类都判断其对象的排序。如果没有该接口，类将不会有这种行为。

有关接口内容将在第 6 章介绍。

### 1.5.6 包

在 Java 中，包用于将相关的类和接口组合起来。它使得一组类仅在需要时才可用，并避免了不同分组的类名可能发生冲突的情形。

默认情况下，您的 Java 类只能访问 java.lang（它提供了基本语言功能，如字符串处理）中的类。要使用其他包中的类，可以通过包名来引用它们或将它们导入到源代码文件中。

要引用包中的类，通常必须使用包的全名。例如，由于类 Color 位于包 java.awt 中，所以在程序中要使用 java.awt.Color 来引用它。

## 1.6 总结

如果您是首次接触面向对象编程，则本章的内容太理论化，您可能一时无法理解。

这是可以理解的，当您的头脑中满是 OOP 概念和术语时，可能会担心无法消化余下的关于 Java 的 20 章内容。

至此，您应对类、对象、属性和行为有基本的了解，同时还应熟悉实例变量和方法。在下一章中，您将使用它们。

有关面向对象编程的其他方面（如继承和包），将在后面的课程中做更详细的介绍。在本书后面的每个课程中，您都将使用面向对象编程。阅读完第一周的课程后，您将拥有使用对象、类、继承以及面向对象编程的其他所有方面的经验。

## 1.7 问与答

问：实际上，方法是在类中定义的函数。既然它们无论从外观和行为方面都类似于函数，为什么不将它们叫做函数呢？

答：有些面向对象编程语言确实将它们叫做函数（C++ 将它们叫做成员函数）。其他一些面向对象语言将位于类（对象）内、外的函数区分开来，因为在这些语言中，使用不同的术语对理解每个函数的工作原理至关重要。因为其他语言有这种区别，同时术语“方法”在面向对象技术中很常用，所以 Java 也使用这个术语。

问：实例变量和实例方法同类变量和类方法之间有何区别？

答：在 Java 程序中，您所做的几乎每项工作涉及的都是实例（也叫做对象）而不是类。然而，对于有些行为和属性，存储在类本身中要比存储在对象中更合理。例如，`java.lang` 包中的 `Math` 类包含一个名为 `PI` 的变量，它存储的是  $\pi$  的近似值。这个值是不变的，因此这个类的不同对象没有必要保留自己的 `PI` 变量。另一方面，每个 `String` 对象都包含了一个 `length()` 方法，它计算该 `String` 中的字符数。这个值对于 `String` 类的每个对象都可能不同，因而它必须是实例方法。

## 1.8 小测验

### 1.8.1 问题

请回答下述 3 个问题，以复习本章介绍的内容。

1. 类又叫做什么？
  - a. 对象；
  - b. 模板；
  - c. 实例。
2. 创建子类时，必须定义它的哪些方面？
  - a. 它已被定义好了；
  - b. 不同于超类的内容；
  - c. 各个方面。
3. 类的实例方法表示的是什么？
  - a. 类的属性；



- b. 类的行为；
- c. 类对象的行为。

答案

1. b, 类是一个抽象模板, 用于创建彼此相似的对象。
2. b, 需要定义子类与超类有什么不同。由于继承的关系, 相同的内容已定义好了。从技术上说, 答案 a 是正确的, 但如果子类中的一切都与超类相同, 就没有必要创建它。
3. c, 实例方法指的是特定对象的行为, 而类方法指的是属于该类的所有对象的行为。

### 1.8.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题, 请回答该问题, 而不要查看本章的内容。

下面的哪些说法是正确的?

- a. 使用同一个类创建的所有对象都必须相同;
- b. 使用同一个类创建的所有对象可以互不相同;
- c. 对象将继承用于创建它的类的属性和行为;
- d. 类将继承其超类的属性和行为。

答案 b

## 1.9 练习

为巩固今天介绍的知识, 请尝试完成下面的练习:

- 在 VolcanoApplication 程序的 main() 方法中, 创建另一个名为 virgiri 的 VolcanoRobot 对象, 设置其实例变量, 并将它们的值显示出来。
- 为国际象棋中的棋子创建一个继承层次结构, 并决定在层次结构的什么位置定义实例变量 color、startingPosition、forwardMovement 和 sideMovement。



## 第 2 章

# Java 编程基础

Java 程序是由类和对象组成的，而对象和类又是由方法和变量组成的。方法是由语句和表达式组成的，表达式又由运算符组成。

至此，您可能担心 Java 就像被称为 matryoshka 的俄罗斯嵌套洋娃娃。每个这样的洋娃娃里边都有一个更小的洋娃娃，而后者同前者一样错综复杂。

本章将消除大洋娃娃的困扰，揭示 Java 编程的最小元素。本章暂时撇开类、对象和方法，介绍单行 Java 代码中的基本元素。

这包括以下内容：

- Java 语句和表达式；
- 变量和基本数据类型；
- 常量；
- 注释；
- 字面量；
- 算术；
- 比较；
- 逻辑运算符。

### 2.1 语句和表达式

您在 Java 程序中要完成的所有任务都可分解为一系列的语句。在编程语言中，语句是简单命令，它导致计算机执行某种操作。

语句表示程序中发生的单个操作。下面的各行代码都是简单的 Java 语句：

```
int weight = 225;

System.out.println("Free the bound periodicals!");

song.duration = 230;
```

有些语句能够提供一个值，例如在将两个数相加或比较两个变量是否相等时。这类语句被称为表达式。

表达式是生成一个值的语句。这个值可以被存储下来，供程序后面使用，也可以立即用于另一条语句中或被丢弃。语句生成的值称为返回值。

有些表达式生成一个数字返回值，例如将两个数相加或相乘时。有些表达式生成一个布尔值（true 或 false）或 Java 对象，这将在本章后面进行介绍。

虽然在很多 Java 程序中，每条语句占一行，但这只是一种格式，并不能决定语句到哪里结束。Java 中的语句都是以分号（;）结尾的。程序员可以在一行中放置多条语句，且它们都能够通过编译，如下所示：

```
dante.speed = 2; dante.temperature = 510;
```



在 Java 中，使用左大括号 ( { ) 和右大括号 ( } ) 对语句进行分组。位于这两个字符之间的语句称为块 (block) 或块语句 (block statement)，这将在第 4 章中做更详细的介绍。

## 2.2 变量和数据类型

在第 1 章中，您创建的应用程序 VolcanoRobot 使用变量来跟踪信息。变量是程序运行时能够存储信息的地方。可在程序的任何地方对其中的值进行修改——因此被称为变量。

要创建变量，必须提供名称并指定它存储的信息类型。还可以在创建变量的同时给它指定初值。

在 Java 中，有 3 种变量。

- 实例变量：正如第 1 章中指出的，实例变量用于定义对象的属性。
- 类变量：定义类的属性，适用于类的所有实例。
- 局部变量：用于方法定义，乃至方法中更小的语句块中。仅当 Java 解释器执行这些方法或语句块时，它们才被使用，离开方法或块之后，它们将不复存在。

虽然这 3 种变量的创建方式极其相似，但在使用方式上，类变量和实例变量不同于局部变量。本章介绍局部变量，而实例变量和类变量将在第 3 章中介绍。

### 注意

与其他语言不同，Java 没有可用于程序的任何地方的全局变量。实例变量和类变量用于在对象间传递信息，因此不需要全局变量。

### 2.2.1 创建变量

在 Java 程序中使用变量之前必须先创建它——声明其名称和存储的信息类型。首先指出信息类型，然后是变量名。下面是一些变量声明的例子：

```
int loanLength;  
String message;  
boolean gameOver;
```

### 注意

变量类型将在本章后面介绍。在上述示例中，int 类型表示整数，String 是用来存储文本的对象，boolean 用来存储 true/false 值。

局部变量可在方法中的任何地方声明，就像其他 Java 语句一样，但必须在使用之前声明它们。变量声明通常位于命名和标识方法的语句之后。

下面的示例在程序的 main() 方法开头定义了 3 个变量：

```
public static void main(String[] arguments) {  
    int total;  
    String reportTitle;  
    boolean active;  
}
```

创建多个相同类型的变量时，可在同一条语句中声明它们，并用逗号将各个变量分隔开来。下面的语句声明了 3 个名为 street、city 和 state 的 String 变量：

```
String street, city, state;
```

在声明变量时，可以使用等号给它赋值。下面的语句创建新变量并给它们指定初值：

```
int zipCode = 90210;  
int box = 350;  
boolean pbs = true;  
String name = "Zoom", city = "Boston", state = "MA";
```

正如最后一条语句表明的，可以使用逗号分隔的方式给多个相同类型的变量赋值。

对于局部变量，在程序中使用它之前，必须对它赋值，否则程序将无法通过编译。因此，良好的习惯是给所有局部变量指定初值。

默认情况下，实例变量和类变量的初值取决于其数据类型。

- 数值变量：0。
- 字符变量：'\0'。
- 布尔变量：false。
- 对象：null。

### 2.2.2 给变量命名

在 Java 中，变量名必须是以字母、下划线（\_）或美元符（\$）开头，而不能以数字开头。在第一个字符之后，变量名可以包含任何字母和数字的组合。

#### 注意

另外，Java 语言使用 Unicode 字符集，该字符集包括标准字符集，外加几千个用于表示国际字母的字符。有 Unicode 字符的重音字符和其他符号也可用于变量名中。

在程序中给变量命名并使用它时，别忘了 Java 是区分大小写的——字母的大小写必须一致。因此，同一个程序中可以有变量 *X* 和 *x*，而 *rose*、*Rose* 和 *ROSE* 是不同的变量。

在本书或其他地方的程序中，Java 变量都被赋予一个有意义的名称，它们由多个单词组合而成。为方便辨识，可使用下述通用规则：

- 变量名的第一个字母小写；
- 变量名中其他单词的第一个字母大写；
- 其他字母都小写。

下面的变量声明遵循了上述命名规则：

```
Button loadFile;
int localAreaCode;
boolean quitGame;
```

### 2.2.3 变量类型

除名称外，变量声明还必须包括存储的信息类型，这可以是：

- 基本数据类型；
- 类名或接口名；
- 数组。

关于如何声明和使用数组变量，将在第 4 章中介绍。本章将重点介绍其他变量类型。

#### 1. 数据类型

基本变量类型有 8 种，它们用于存储整数、浮点数、字符和布尔值。它们通常被称为简单类型（primitive type），因为它们是 Java 内置的而不是对象，这使得它们使用起来效率更高。不管在什么操作系统和平台上，这些数据类型的长度和特征都相同，这与其他编程语言中的某些数据类型不同。

用于存储整数的数据类型有 4 种，如表 2.1 中所示。具体使用哪种类型取决于要存储的整数的大小。

表 2.1 整 型

类型	大小 (位)	取值范围
byte	8	-128 ~ 127
short	16	-32 768 ~ 32 767
int	32	-2 147 483 648 ~ 2 147 483 647
long	64	-9 223 372 036 854 775 808 ~ 9 223 372 036 854 775 807

所有这些类型都是有符号的,这意味着它们既可以存储正数,也可以存储负数。给变量指定哪种类型取决于它需要存储的值的范围。所有这些整型变量都不能正确存储超出其取值范围的值,所以在指定类型时一定要小心。

另一种数值是浮点数,其类型为 float 或 double。浮点数是带小数的数字。大多数情况下, float 类型已经足够了,因为其取值范围为  $1.4\text{E}-45$  到  $2.4\text{E}+38$ ;如果还不够,可使用 double 类型,其取值范围为  $4.9\text{E}-324$  到  $1.7\text{E}+308$ 。

char 类型用于存储单个字符,如字母、数字、标点和其他符号。

最后一种基本类型是 boolean。正如前面介绍过的,其取值为 true 或 false。

所有这些变量类型名都是小写的,在程序中必须这样使用它们。存在与这些数据类型名称相同,但大小写不同的类,如 Boolean 和 Char。在 Java 程序中,它们的功能不同,不能互换。下一章将介绍如何使用这些特殊的类。

**注意**

如果算上 void, Java 中实际上有 9 种基本类型。void 表示“空”(nothing),用于指出方法不返回任何值。

**2. 类的类型**

除基本数据类型外,变量的类型还可以是类,如下所示:

```
String lastName = "Hopper";

Color hair;

VolcanoRobot vr;
```

当变量的类型为类时,它指的是这种类或其子类的一个对象。

在上述代码中, VolcanoRobot vr; 声明一个名为 vr 的变量,用于存储一个 VolcanoRobot 对象,虽然对象本身可能还不存在。下一章将介绍如何将对象和变量关联起来。

当变量可能是几个子类之一时,将其类型声明为这些子类的超类很有用。例如,假设在某个类层次结构中,有超类 CommandButton 和 3 个子类——RadioButton、CheckboxButton 和 ClickButton,则创建一个名为 widget 的 CommandButton 变量后,它可用于存储 RadioButton、CheckboxButton 或 ClickButton 对象。

Object 类型的变量可以与任何类型的对象关联在一起。

**2.2.4 给变量赋值**

声明变量后,可以用赋值运算符(等号,=)给它赋值。下面是两个赋值语句的例子:

```
idCode = 8675309;

accountOverdrawn = false;
```

### 2.2.5 常量

如果需要存储在程序运行时可以修改的信息，使用变量很有用。

对于在程序运行过程中一直不变的值，可以用一种特殊的变量——常量。常量也称为常数变量 (constant variable)，是值保持不变的变量。这里的“变量”有些用词不当。

对于为对象的所有方法定义共享值而言，常量很有用。在 Java 中，可以创建各种类型的常量：实例常量、类常量和局部常量。

要声明常量，可在变量声明前加上关键字 `final`，并给变量指定一个初值，如下所示：

```
final float PI = 3.141592;

final boolean DEBUG = false;

final int PENALTY = 25;
```

在上述语句中，常量名都为大写：PI、DEBUG 和 PENALTY。并非必须这样做，但很多 Java 程序员都这样做，以表明这是常量而不是变量。

可使用常量来表示对象的不同状态，然后测试这些状态。假设有一个这样的程序，它直接从数字键盘读取方向输入——8 表示向上，4 表示向左等，则您可以将这些值定义为 `int` 常量：

```
final int LEFT = 4;
final int RIGHT = 6;
final int UP = 8;
final int DOWN = 2;
```

常量让程序更容易理解。为说明这一点，请看下面哪条语句更清楚地说明了其功能：

```
guide.direction = 4;

guide.direction = LEFT;
```

本章要介绍的第一个项目是一个这样的 Java 应用程序：创建多个变量，给它们赋初值，然后显示其中两个变量的值，如程序清单 2.1 所示。

程序清单 2.1 Variables.java 的源代码

---

```
1: public class Variables {
2:
3:     public static void main(String[] arguments) {
4:         final char UP = 'U';
5:         byte initialLevel = 12;
6:         short location = 13250;
7:         int score = 3500100;
8:         boolean newGame = true;
9:
10:        System.out.println("Level: " + initialLevel);
11:        System.out.println("Up: " + UP);
12:    }
13: }
```

---

编译该应用程序并运行类文件 `Variable.class`，其输出如下：

```
Level: 12
Up: U
```

这个类使用了 4 个局部变量和 1 个常量，并在第 10~11 行使用 `System.out.println()` 来生成输出。

`System.out.println()` 是一个用于将字符串和其他信息显示到标准输出设备（通常是屏幕）上的方法。

`System.out.println()` 接收一个参数：一个字符串。要将多个变量或字面值作为 `println()` 的参数，可以使用运算符 “+” 将它们合并成一个字符串，这将在本章后面介绍。

还有方法 `System.out.print()`，它在字符串后加上回车。要在同一行显示多个字符串，可连续调用方法 `print()`，而不是 `println()`。

## 2.3 注释

为提高程序的可读性，最重要的方式之一是使用注释。注释是程序中，旨在帮助了解程序的功能的信息。Java 编译器在生成 Java 源代码文件的可执行版本时将忽略注释。

在 Java 程序中，可以使用 3 种不同类型的注释，具体使用哪一种完全取决于您自己。第一种添加注释的方式是，以两个斜杠 (//) 打头。斜杠到行尾的所有内容都是注释，Java 编译器将忽略它们，如下面的语句所示：

```
int creditHours = 3; // set up credit hours for course
```

如果需要编写多行的注释，可以以 /\* 打头，并以 \*/ 结束。这两个分界符之间的所有内容都被视为注释，如下所示：

```
/* This program occasionally deletes all files on  
your hard drive and renders it completely unusable  
when you press the Save button. */
```

最后一种注释类型既供程序员阅读，也供计算机阅读。以 /\*\* (而不是 /\*) 打头，并以 \*/ 结束的注释将被视为用于描述类及其公有方法的正式文档。

这种注释可被诸如 JDK 中的 javadoc 工具实用程序读取。Javadoc 程序使用这种注释来创建一组超文本标记语言 (HTML) 文件，用于说明该程序及其类层次结构和方法。有关 javadoc 的更详细的信息，请参阅附录 B。

### 提示

Java 类库中的所有正式文档都来自 javadoc 风格的注释。

## 2.4 字面量

除变量外，还可以在 Java 语句中使用字面量。字面量可以是任何直接表示一个值的数字、文本或其他信息。

下面的赋值语句使用了字面量：

```
int year = 2007;
```

其中的字面量是 2007，因为它直接表示整数值 2007。数字、字符和字符串都是字面量。

虽然在大多数情况下，字面量的用法和含义是显而易见的，但 Java 中有些特殊类型的字面量，它们表示各种数字、字符、字符串和布尔值。

### 2.4.1 数字字面量

Java 有几种整型字面量。例如，数字 4 是一个 int 类型的整型字面量，可以被赋给 byte 或 short 类型的变量，因为它足够小，位于这些整数类型的取值范围之内。int 的取值范围之外的整型字面量将被视为 long 类型。也可以在后面加上字母 L (或 l) 来指出字面量的类型为 long。例如，下面的语句将 4 视为一个 long 值：

```
pennyTotal = pennyTotal + 4L;
```

要表示负的数字字面量，可在前面加上负号 (-)，如 -45。

### 注意

Java 也支持使用八进制和十六进制表示的数字字面量。

八进制是以 8 为基数的计数系统，这意味着每位只能是 0 和 7 之间的值。在八进制数中，第 8 个数是 10 (Java 字面量为 010)。

十六进制是以 16 为基数的计数系统，每位可能的取值为 16 个。字母 A~F 表示最

后的 6 个数字，因此这 16 个数依次为 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。

对于有些编程任务而言，八进制和十六进制系统比十进制系统更适合。如果您曾经使用过 HTML 来设置网页的背景颜色，则可能使用过十六进制数。

如果需要使用八进制的字面量整数，可在前面加上 0。例如，八进制的 777 应为 0777。对于十六进制的整数字面量，应在前面加上 0x，如 0x12 或 0xFF。

浮点数字面量使用句点 (.) 来表示小数点。下面的语句使用字面量来设置一个 double 变量的值：

```
double myGPA = 2.25;
```

所有的浮点数字面量都被视为 double 类型，而不是 float 类型。要将字面量的类型指定为 float，可加上字母 F (或 f)，如下所示：

```
float piValue = 3.1415927F;
```

在浮点数字面量中，可以使用指数表示法，即使用字母 e (或 E)，指数可以是负数。下面的语句使用了指数表示法：

```
double x = 12e22;
```

```
double y = 19E-95;
```

## 2.4.2 布尔字面量

布尔值 true 和 false 也是字面量。boolean 变量的取值只能是 true 或 false。

下面的语句给一个 boolean 变量赋值：

```
boolean chosen = true;
```

如果您使用过其他语言，如 C 语言，则可能认为 1 和 true 等价，而 0 和 false 等价。在 Java 中，情况并非如此，必须使用值 true 和 false 来表示布尔值。

注意字面量 true 并不需要用引号括起。如果被引号括起，则 Java 编译器会将其视为一个字符串。

## 2.4.3 字符字面量

字符字面量是用单引号括起的单个字符，如 'a'、'#' 和 '3'。您也许熟悉 ASCII 字符集，它包括 128 个字符，其中有数字、字母、标点和其他对计算有帮助的符号。Java 使用 16 位的 Unicode 标准，支持数以千计的其他字符。

一些字符字面量表示的是非打印字符或不能通过键盘输入的字符。表 2.2 列出了一些特殊编码，它们用于表示这些特殊字符以及 Unicode 字符集中的字符。在八进制、十六进制和 Unicode 转义编码中，字母 d 表示一个数字或十六进制数字 (a~f 或 A~F)。

表 2.2 字符的转义编码

转义符	含义
\n	换行
\t	水平制表符
\b	退格
\r	回车
\f	换页
\\	反斜杠
'\'	单引号

续表

转义符	含义
\"	双引号
\d	八进制
\xd	十六进制
\ud	Unicode 字符

### 2.4.4 字符串字面量

Java 程序可能使用的最后一种字面量表示一个字符串。Java 中的字符串是一种对象，而不是一种基本数据类型。同时，不像 C 语言那样，字符串被存储在数组中。

因为字符串对象是 Java 中的真正对象，所以存在用于合并和修改字符串以及判断两个字符串是否相同的方法。

字符串字面量是用双引号括起的一系列字符，如下所示：

```
String quitMsg = "Are you sure you want to quit?";
```

```
String password = "swordfish";
```

字符串中可以包含表 2.2 列出的转义字符，如下所示：

```
String example = "Socrates asked, \"Hemlock is poison?\"";
```

```
System.out.println("Sincerely,\nMillard Fillmore\n");
```

```
String title = "Sams Teach Yourself Ruby on Rails While You Sleep\u2122";
```

在上述最后一行代码中，在支持 Unicode 的系统上，Unicode 编码序列 \u2122 将生成一个™符号。

**警告**

虽然 Java 支持对 Unicode 字符的传输，但要显示这些字符，用户的系统也必须支持 Unicode。Unicode 提供了一种对字符进行编码的方式，可用于支持该标准的系统。Java 2 支持任何 Unicode 字符的显示，只要该字符能够被主机的某种字体表示出来。

有关 Unicode 的更详细的信息，请访问 Unicode 联盟的网站 <http://www.unicode.org>。

虽然在程序中使用字符串字面量的方式与其他字面量类似，但在后台对它们的处理是不一样的。

对于字符串字面量，Java 将其存储为 String 对象。您不必像使用其他对象那样，显式地创建一个新对象，因此使用起来与基本数据类型一样简单。从这种意义上说，字符串与众不同——基本数据类型都不会被存储为对象。本章后面和下一章将更详细地介绍字符串和 String 类。

## 2.5 表达式和运算符

表达式是一条能够提供值的语句。最常见的是数学表达式，如下面的例子所示：

```
int x = 3;
int y = x;
int z = x * y;
```

这 3 条语句都是表达式——它们提供了可被赋给变量的值。第 1 条语句将字面量 3 赋给变量 x。第 2 条语句将变量 x 的值赋给变量 y。在第 3 条语句中，乘法运算符 “\*” 用来将 x 和 y 相乘，结果将存储在变量 z 中。

表达式可以是任何变量、字面量和运算符的组合，也可以是方法调用，因为方法能够将一个值返回给调用它的类或对象。

您知道，表达式所提供的值称为返回值。在 Java 程序中，可将这个值赋给变量或以其他方式使用。

大多数 Java 中的表达式使用了运算符，如\*。运算符是一些特殊符号，用于数学函数、赋值语句和逻辑比较。

### 2.5.1 算术运算符

在 Java 中，有 5 种用来完成基本算术运算的运算符，如表 2.3 所示。

表 2.3 算术运算符

运算符	含义	例子
+	加	3+4
-	减	5-7
*	乘	5*5
/	除	14/7
%	求模	20%7

每个运算符都有两个操作数，一边一个。减法运算符也可以用来对单个操作数求反，即将操作数与-1 相乘。

使用除法时，要注意操作数的类型。如果将除法运算的结果存储在整型变量中，结果将被向下取整，因为 int 类型不能处理浮点数。例如，如果将结果存储在 int 变量中，则表达式 31/9 的结果是 3。

使用运算符%的求模运算得到的是除法运算的余数。例如，31%9 的结果是 4，因为 31 被 9 除的余数是 4。

很多针对整数的算术运算的结果为 int 值，而不管操作数是什么类型。处理其他数值类型，如浮点数或长整型时，应确保操作数的类型与所需结果的类型相同。

程序清单 2.2 是一个简单的算术运算示例。

程序清单 2.2 源代码文件 Weather.java

```

1: public class Weather {
2:     public static void main(String[] arguments) {
3:         float fah = 86;
4:         System.out.println(fah + " degrees Fahrenheit is ...");
5:         // To convert Fahrenheit into Celsius
6:         // Begin by subtracting 32
7:         fah = fah - 32;
8:         // Divide the answer by 9
9:         fah = fah / 9;
10:        // Multiply that answer by 5
11:        fah = fah * 5;
12:        System.out.println(fah + " degrees Celsius\n");
13:
14:        float cel = 33;
15:        System.out.println(cel + " degrees Celsius is ...");
16:        // To convert Fahrenheit into Celsius
17:        // Begin by subtracting 32
18:        cel = cel * 9;
19:        // Divide the answer by 9
20:        cel = cel / 5;
21:        // Multiply that answer by 5
22:        cel = cel + 32;
23:        System.out.println(cel + " degrees Fahrenheit");
24:    }
25: }

```

运行该 Java 应用程序，其输出如下：



```
86.0 degrees Fahrenheit is ...
30.0 degrees Celsius

33.0 degrees Celsius is ...
91.4 degrees Fahrenheit
```

该 Java 应用程序的第 3~12 行使用算数运算符将华氏温度转换为摄氏温度：

- 第 3 行：声明浮点变量 *fah*，并将其初始化为 86。
- 第 4 行：显示 *fah* 的当前值。
- 第 5 行：第一个注释，它解释了该程序的功能。这些注释将被 Java 编译器忽略。
- 第 7 行：将 *fah* 的值减去 32。
- 第 9 行：将 *fah* 的值除以 9。
- 第 11 行：将 *fah* 的值乘以 5。
- 第 12 行：将 *fah* 的值转换为摄氏温度后，再次显示它。

第 14~23 行与此类似，但执行的是相反的转变——将摄氏温度转换为华氏温度。

## 2.5.2 再谈赋值

给变量赋值是一个表达式，因为它生成一个值。可以这样将赋值语句连在一起：

```
x = y = z = 7;
```

在这条语句中，所有 3 个变量的值最后都为 7。

在赋值之前，将首先计算赋值表达式右边。因此，可以这样使用表达式语句：

```
int x = 5;
x = x + 2;
```

在表达式  $x = x + 2$  中，首先计算  $x + 2$ ；然后将结果（7）赋给变量 *x*。

在编程中，使用表达式来修改变量的值是一项常见的任务。有几个专门用于这方面的运算符。

表 2.4 列出了这些赋值运算符及等效的表达式。

表 2.4 赋值运算符

表达式	含义
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$

### 警告

这些简化的赋值运算符与它们所代表的更长的赋值语句等效。然而，如果赋值语句的一边是复杂的表达式，将出现不再等效的情况。例如，如果 *x* 等于 20，*y* 等于 5，下面两条语句的结果将不同：

```
x = x / y + 5;
x /= y + 5;
```

有疑问时，应使用多条赋值语句来简化表达式，而不要使用简化的运算符。

## 2.5.3 递增和递减运算符

另一种常见的编程任务将整型变量加 1 或减 1。有专门用于完成这种运算的运算符：递增和递减运算符。对变量做递增运算意味着将它的值加 1，做递减运算意味着将它的值减 1。

递增运算符是`++`，递减运算符是`--`。这些运算符位于变量名的前面或后面，如下所示：

```
int x = 7;
x = x++;
```

其中语句 `x = x++` 将变量 `x` 的值从 7 增加到 8。

递增和递减运算符可放在变量名的前面或后面，这将影响表达式的结果。

如果递增和递减运算符位于变量名前面，则被称为前缀运算符；如果位于变量名后面，则被称为后缀运算符。

在简单表达式（如 `standards--`；）中，无论使用前缀还是后缀运算符，结果都相同，因此它们可以互换。然而，当递增和递减运算符被用于更复杂的表达式中时，选择前缀还是后缀运算符就至关重要。

请看下面的代码：

```
int x, y, z;
x = 42;
y = x++;
z = ++x;
```

其中的 3 个表达式由于前缀和后缀运算符的差别而产生完全不同的结果。

使用后缀运算符时，如 `y = x++` 中，先将 `x` 的值赋给 `y`，然后将 `x` 的值加 1；使用前缀运算符时，如在 `z = ++x` 中，`x` 的值先加 1，然后再被赋值给 `z`。最终结果是，`y` 等于 42，`z` 等于 44，`x` 等于 44。

如果对此还不清楚，下面的例子再次通过注释描述了每一步的情况：

```
int x, y, z; // x, y, and z are all declared
x = 42;      // x is given the value of 42
y = x++;     // y is given x's value (42) before it is incremented
              // and x is then incremented to 43
z = ++x;     // x is incremented to 44, and z is given x's value
```

#### 警告

与简化运算符一样，在特别复杂的表达式中使用递增和递减运算符时，很可能得到意外的结果。“`x` 自增之前将 `x` 的值赋给 `y`”这一概念不再完全正确，因为 Java 总是先计算表达式右边的内容，然后再把结果赋给表达式左边的变量。Java 在处理表达式之前会存储一些值，以便后缀运算符能够像本节介绍的那样工作。当包含前缀和后缀运算符的复杂表达式的结果不同于您的期望时，请将该表达式拆成多个语句，以简化它。

### 2.5.4 比较运算符

Java 有几种可用于对变量之间、变量和字面量（或其他类型的信息）之间进行比较的运算符。

这些运算符用于返回布尔值（`true` 或 `false`）的表达式，表达式的返回值取决于比较的结果是真还是假。表 2.5 列出了这些比较运算符。

表 2.5 比较运算符

运算符	含义	例子
<code>==</code>	相等	<code>x==3</code>
<code>!=</code>	不等	<code>x!=3</code>
<code>&lt;</code>	小于	<code>x&lt;3</code>
<code>&gt;</code>	大于	<code>x&gt;3</code>
<code>&lt;=</code>	小于等于	<code>x&lt;=3</code>
<code>&gt;=</code>	大于等于	<code>x&gt;=3</code>

下面的例子演示了比较运算符的用法：

```
boolean hip;
int age = 36;
hip = age < 25;
```

表达式 `age < 25` 的结果为 `true` 还是 `false` 取决于 `age` 的值。由于这里的 `age` 为 36（这比 25 大），因此 `hip` 被赋给布尔值 `false`。

### 2.5.5 逻辑运算符

结果为布尔值的表达式（如比较运算）可以被组合成更加复杂的表达式。这是通过逻辑运算符来实现的，逻辑运算符用于逻辑组合：AND、OR、XOR 和逻辑 NOT。

对于 AND 组合，可使用逻辑运算符 `&` 或 `&&`。当两个布尔表达式通过 `&` 或 `&&` 被组合在一起后，仅当两个布尔表达式都为真时，整个表达式的结果才为 `true`。

请看下面的例子：

```
boolean extraLife = (score > 75000) & (playerLives < 10);
```

这个表达式将两个比较表达式——`score > 75000` 和 `playerLives < 10`——组合在一起。如果这两个表达式都为真，则 `true` 被赋给变量 `extraLife`；在其他情况下，`false` 被赋给 `extraLife`。

`&` 和 `&&` 之间的差别在于 Java 对组合表达式所做的工作量。如果使用 `&`，则不管什么情况下，& 两边的表达式都将被计算；如果使用 `&&`，则当左边的表达式为 `false` 时，将不计算右边的表达式。

对于 OR 组合，可使用逻辑运算符 `|` 或 `||`。如果运算符两边的任何一个布尔表达式为真，则组合表达式的结果为 `true`。

请看下述示例：

```
boolean extraLife = (score > 75000) || (playerLevel == 0);
```

该表达式合并了两个比较表达式：`score > 75000` 和 `playerLevel == 0`。如果这两个表达式中的任何一个为真，则 `true` 被赋给变量 `extraLife`；仅当这两个表达式都是 `false` 时，`false` 才被赋给 `extraLife`。

注意，这里使用的是 `||`，而不是 `|`。因此，如果 `score > 75000` 为 `true`，则 `extraLife` 将被设置成 `true`，而第二个表达式不会被计算。

用于 XOR 合并的逻辑运算符只有一个：`^`。仅当被合并的两个布尔表达式的值相反时，整个表达式的结果才为 `true`；如果两个表达式都是 `true` 或都是 `false`，则 `^` 运算符的结果将为 `false`。

NOT 组合使用逻辑运算符 `!`，后面跟一个表达式。它将对布尔表达式的值求反，这与用负号来改变数字的符号相同。

例如，如果 `age < 30` 的结果为 `true`，则 `!(age < 30)` 的结果将为 `false`。

首次见到这些逻辑运算符时，可能觉得它们完全不符合逻辑。在以后的章节中，您将大量地使用它们，尤其是在第 5 章中。

### 2.5.6 运算符优先级

当表达式中有多个运算符时，Java 有一套优先级用来判断运算符的执行顺序。在许多情况下，优先级决定了整个表达式的最终值。

例如，请看下面的表达式：

```
y = 6 + 4 / 2;
```

变量 `y` 的值为 5 还是 8 取决于哪个算术运算先被处理。如果先计算表达式 `6+4`，则 `y` 的值为 5；否则为 8。

通常，从先到后的顺序如下：

- 递增和递减运算；
- 算术运算；

- 比较运算；
- 逻辑运算；
- 赋值运算。

如果两个运算的优先级相同，则左边的比右边的先被处理。表 2.6 列出了 Java 中众多运算符的优先级。排在越前面的运算符的优先级越高。

表 2.6 运算符优先级

运算符	说明
<code>() [] 0</code>	圆括号 <code>()</code> 用来将表达式分组；句点 <code>.</code> 用来访问对象和类中的方法和变量（这将在下一章讨论）；方括号 <code>[]</code> 用于数组（这个运算符将在本周后面讨论）
<code>++ -- ! ~ instanceof</code>	<code>instanceof</code> 运算符返回 <code>true</code> 或 <code>false</code> 值，这取决于该对象是否属于指定的类或其子类的一个实例（这将在下一章讨论）
<code>New(type)expression</code>	<code>new</code> 运算符用来创建类的新实例，其中 <code>()</code> 用于将值转换为另一种类型（这两种运算符都将在下一章进行讨论）
<code>* / %</code>	乘法、除法和求模运算
<code>+ -</code>	加减
<code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code>	按位左移和右移
<code>&lt; &gt; &lt;= &gt;=</code>	关系比较
<code>= !=</code>	相等和不等
<code>&amp;</code>	AND
<code>^</code>	XOR
<code> </code>	OR
<code>&amp;&amp;</code>	逻辑与
<code>  </code>	逻辑或
<code>?:</code>	<code>If...then...else</code> 的简写（将在第 4 章中讨论）
<code>= += -= *= /= %= ^=</code>	各种赋值运算
<code>&amp;=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</code>	其他赋值运算

回过头来看表达式  $y = 6 + 4 / 2$ 。表 2.6 表明，除法先于加法计算，因此  $y$  的值应为 8。

要改变表达式求值的顺序，可以用圆括号将需要先计算的表达式括起。可以嵌套圆括号以确保按所需的顺序对表达式进行求值——最里边的圆括号内的表达式最先被计算。

下述表达式的结果为 5：

```
y = (6 + 4) / 2
```

因为先计算  $6 + 4$ ，得到 10，然后再除以 2，最后的结果为 5。

圆括号还可以提高表达式的可读性。如果不能一眼看出表达式的优先级，可以添加圆括号来强制转换为希望的优先级，使语句更容易理解。

## 2.6 字符串运算

正如本章前面指出的，在数学领域之外，运算符“+”有另一种功能，用于拼接两个或更多的字符串。拼接（concatenate）指的是将两样东西连接到一起。不知道什么原因，选择这个动词来描述合并字符串的操作——它从 `paste`、`glue`、`affix`、`combine`、`link` 和 `conjoin` 等中脱颖而出。

在一些例子中，您已经看到了像下面这样的语句：

```
String firstName = "Raymond";
System.out.println("Everybody loves " + firstName);
```

这两行代码将显示下述文本：

```
Everybody loves Raymond
```

运算符“+”将字符串、其他对象和变量合并为一个字符串。在上面的例子中，字面量 `Everybody loves` 与 `String` 对象 `firstName` 的值合并在一起。

在 Java 中，拼接运算符的用法很简单，因为该字符串将任何变量类型和对象值都作为字符串进行处理。如果拼接运算的任何一个部分是 `String` 或 `String` 字面量，则其他的所有元素都将被作为字符串进行处理：

```
System.out.println(4 + " score and " + 7 + " years ago");
```

这将产生文本输出 `4 score and 7 years ago.`，就像整数字面量 `4` 和 `7` 是字符串一样。

也可以用简化运算符 `+=` 来在字符串末尾添加内容。例如，请看下面的表达式：

```
myName += " Jr.";
```

这个表达式等效于：

```
myName = myName + " Jr.";
```

在这个例子中，它修改了 `myName` 的值（原来可能是 `Efrem Zimbalist`）——在后面添加 `Jr.`（因此变成 `Efrem Zimbalist Jr.`）。

## 2.7 总结

任何打开一套 `matryoska` 洋娃娃的人，在找到最小的洋娃娃后，都难免会失望。理想情况下，随着微工程技术的进展，俄罗斯的工匠们将能够制作出越来越小的洋娃娃，直到达到亚原子极限，才算是胜利者。

在本章中，您看到了 Java 中最小的洋娃娃，但它应该不会让人失望。语句和表达式让您能够创建高效的方法，进而创建出高效的对象和类。

在本章中，您学习了如何创建变量并给它赋值，还学习了使用字面量来表示数字、字符和字符串值以及如何使用运算符。在下一章中，您将使用这些技术来开发类。

表 2.7 列出了本章介绍的运算符，以对本章的内容做一总结。请将它视为一个洋娃娃，仔细地查看。

表 2.7 运算符小结

运算符	含义
+	加
-	减
*	乘
/	除
%	求模
<	小于
>	大于
<=	小于等于
>=	大于等于
=	相等
!=	不等
&&	逻辑与
	逻辑或
!	逻辑非

续表

运算符	含义
&	与
	或
^	异或
=	赋值
++	递增
--	递减
+=	相加然后赋值
-=	相减然后赋值
*=	相乘然后赋值
/=	相除然后赋值
%=	求模然后赋值

## 2.8 问与答

问：如果将一个超出变量取值范围的整数值赋给该变量，将发生什么情况？

答：从逻辑上说，您可能认为该变量将被转换为与之接近的更大类型，但情况并非如此。相反，将发生溢出，即从一个极端回到另一个极端。例如，*byte* 变量的值从 127（可接受的值）变到 128（不可接受）时，将转到最小的可接受值，即 -128，然后往上增大。您并不希望程序中发生溢出，因此将值赋给变量时，不应超过其所属数据类型的取值范围。

问：为什么 **Java** 包含所有对应于数学运算和赋值的简化运算符？它们不太好阅读。

答：Java 的语法是基于 C++ 的，而后者又是基于 C 的（又是一个俄罗斯嵌套洋娃娃）。C 是一种专家语言，它更重视功能，而不是可读性，简化运算符是这种设计思想的产物之一。并不是非得在程序中使用它们，因为可能采用其他方式。如果愿意，可以在程序中尽量避免使用它们。

## 2.9 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 2.9.1 问题

- 下面哪个是合法的布尔值？
  - "false";
  - false;
  - 10。
- 下面哪条不属于 Java 的变量命名约定？
  - 除第一个单词外，变量名中其他单词的首字母都应大写；
  - 变量名的第一个字母小写；
  - 所有字母都大写。
- 下面哪种数据类型的取值范围为 -32 768~32 767？
  - char;



- b. byte,
- c. short。

答案

1. b, 在 Java 中, 布尔值只能是 true 或 false。用引号将这些值括起时, 它们将被视为 String, 而不是布尔值。
2. c, 只有常量名大写, 以便将其同变量区分开来。
3. c,

### 2.9.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题, 请回答该问题, 而不要查看本章的内容。

下面哪组数据类型能够存储 3000000000?

- a. short、int、long、float,
- b. int、long、float,
- c. long、float,
- d. byte。

答案 c

### 2.10 练习

为巩固本章介绍的知识, 请尝试完成下面的练习。

- 创建一个程序, 计算\$14000 的投资在三年后值多少。假设第一年增值 40%, 第二年损失\$1500, 第三年又增值 12%。
- 编写一个程序, 它显示两个数字, 并使用/和%来显示它们相除后的商和余数。输出时, 使用转义符\t 来将商和余数分开。



## 第 3 章

# 对 象

Java 是一种面向对象性非常强的编程语言。在 Java 中，您使用对象来完成任务。您创建对象，修改和移动它们，修改其变量，调用其方法，将它们与其他对象合并起来。您开发类，使用这些类来创建对象，并将它们与其他类和对象一起使用。

本章将大量地使用对象，这包括如下主题：

- 创建对象（也叫做实例）；
- 测试和修改对象的类变量和实例变量；
- 调用对象的方法；
- 将对象和其他数据类型从一个类转换为另一个类。

### 3.1 创建新对象

编写 Java 程序时，您定义一系列的类。正如第 1 章中介绍的，类是用来创建对象的模板。对象（也叫实例）是程序的自包含元素，它们包含相关的特性和数据。从很大程度上说，您是用类创建实例，然后对实例进行操作。因此本节将介绍如何使用类来创建新的对象。

还记得上一章介绍的字符串吗？使用字符串字面量（用双引号括起的字符序列）可以创建一个新的 String 类实例，该实例的值为该字符串。

从这种意义上说，String 类与众不同。虽然它也是一个类，但有一种简单的方法来创建这种类的实例，即使用字面量。要创建其他类的实例，需要使用 new 运算符。

#### 注意

数字和字符串字面量又如何呢？它们也会创建对象吗？不会。数字和字符串基本数据类型创建数字和字符串，但为提高效率，它们不是对象。如果需要将它们当作对象处理，可以将它们包裹成对象（这将在第 5 章中介绍）。

#### 3.1.1 使用 new

要创建新对象，可以使用 new 运算符和要创建的对象所属的类的名称，并加上圆括号，如下面 3 个例子所示：

```
String name = new String();
URL address = new URL("http://www.java21days.com");
VolcanoRobot robbie = new VolcanoRobot();
```

圆括号很重要，不可缺少。括号里可以为空，在这种情况下，创建的将是最简单、最基本的对象，也可以包含参数，这些参数决定了对象的实例变量的初始值和其他初始量。

下面的例子演示如何在创建对象时提供参数：

```
Random seed = new Random(6068430714);
```



```
Point pt = new Point(0, 0);
```

圆括号中可包含的参数个数和类型是由类本身决定的，这是通过一种叫做构造函数（constructor）的特殊方法来定义的（本章后面将更详细地介绍构造函数）。如果您使用类创建对象时，提供的参数的数目和类型不正确（或者在需要参数时，您没有提供），则编译该 Java 程序时将出错。

下面是一个使用不同数目和类型的参数创建不同类型对象的例子：StringTokenizer 类。这个类位于 java.util 包中，它将一个字符串划分为一系列叫做标记（token）的字符串。

可以通过将某个字符或多个字符作为分界符，来将字符串划分为多个标记。例如，使用斜杠字符 (/) 作为分界符时，文本“02/20/67”可被分为 3 个标记：02、20 和 67。

程序清单 3.1 中的 Java 程序使用 new 以两种不同的方法创建 StringTokenizer 对象，然后显示每个对象包含的标记。

程序清单 3.1 完整的 TokenTester.java 源代码

```
1: import java.util.StringTokenizer;
2:
3: class TokenTester {
4:
5:     public static void main(String[] arguments) {
6:         StringTokenizer st1, st2;
7:
8:         String quote1 = "VIZY 3 -1/16";
9:         st1 = new StringTokenizer(quote1);
10:        System.out.println("Token 1: " + st1.nextToken());
11:        System.out.println("Token 2: " + st1.nextToken());
12:        System.out.println("Token 3: " + st1.nextToken());
13:
14:        String quote2 = "NPLI@9 27/32@3/32";
15:        st2 = new StringTokenizer(quote2, "@");
16:        System.out.println("\nToken 1: " + st2.nextToken());
17:        System.out.println("Token 2: " + st2.nextToken());
18:        System.out.println("Token 3: " + st2.nextToken());
19:    }
20: }
```

编译并运行该程序时，其输出如下：

```
Token 1: VIZY
Token 2: 3
Token 3: -1/16
```

```
Token 1: NPLI
Token 2: 9 27/32
Token 3: 3/32
```

在这个例子中，创建了两个不同的 StringTokenizer 对象，这是通过在 new 后面的构造函数中提供不同的参数来实现的。

第一个实例（第 9 行）是使用一个参数（名为 quote1 的 String 对象）创建的。这创建了一个使用默认分界符的 StringTokenizer 对象。默认分界符包括空格、水平制表符、换行符、回车或换页字符。

如果字符串中包含其中任何一种字符，该字符都将被用来划分标记。由于字符串 quote1 中包含空格，所以将使用空格来划分标记。第 10~12 行显示了所有的 3 个标记：VIZY、3 和 -1/16。

第 14 行创建第二个 StringTokenizer 对象时，提供了两个参数：String 对象 quote2 和字符 @。第二个参数指定将字符 @ 作为标记间的分界符。第 15 行创建的 StringTokenizer 对象包括 3 个标记：NPLI、9 27/32 和 3/32。

#### 注意

在第 18 章的 QuoteData 项目中，您将使用 Yahoo Finance 的实时股票数据。该应用程序使用 StringTokenizer 来分离用逗号分隔的股票报价字段。

### 3.1.2 new 的功能

当您使用运算符 `new` 时，将发生如下几件事：创建给定类的实例，为它分配内存，调用给定类定义的一个特殊方法。该特殊方法叫做构造函数。

构造函数是专门用于创建和初始化类实例的方法。构造函数初始化新对象及其变量，创建该对象所需的其他对象，并执行初始化该对象所需的其他操作。

在同一个类中，可以定义多个构造函数，其中每个构造函数的参数数目和类型各不相同。使用 `new` 时，您可以在参数列表中指定不同的参数，这样将调用相应的构造函数。在前面例子中，`TokenTester` 类之所以能够通过以不同的方式使用 `new` 运算符来完成不同的工作，是因为定义了多个构造函数。创建自己的类时，您可以根据需要定义任意数目的构造函数，以实现类的行为。

### 3.1.3 内存管理

如果您熟悉其他面向对象编程语言，也许会问：是否有一个与 `new` 对应的语句，用于在对象不再需要时将其释放？

在 Java 中，内存管理是动态的、自动的。当您创建新对象时，Java 自动为该对象分配适当数量的内存，您不必显式地为对象分配内存。这项工作由 Java 完成。

由于内存管理是自动的，所以使用完对象后，您不必释放该对象占用的内存。在大多数情况下，当您使用完对象后，Java 能够判断出该对象已经不再有任何活动的引用（即该对象没有被赋给任何正在使用或被存储数组中的变量）。

程序运行时，Java 定期地查询未使用的对象，并收回这些对象占用的内存。这被称为无用单元收集（garbage collection），它是完全自动的。您不必显式地释放对象占用的内存；您只需确保不再占用要删除的对象。

## 3.2 访问和设置类变量和实例变量

至此，您能够创建包含类变量和实例变量的对象，但如何使用这些变量呢？这很容易！类变量和实例变量的用法在很大程度上与上一章介绍的局部变量相同。您可以将它们用于表达式中，在语句中给它们赋值，等等。只是在代码中引用它们的方式与引用常规变量稍有不同。

### 3.2.1 获取值

要获取实例变量的值，可以使用句点表示法，实例变量和类变量由两个部分组成：句点左边为对象和类的引用，句点右边为变量。

句点表示法是一种使用句点（`.`）运算符来引用对象的实例变量和方法的方式。

例如，如果有名为 `myCustomer` 的对象，而该对象有一个名为 `orderTotal` 的变量，则可以这样引用该变量：

```
float total = myCustomer.orderTotal;
```

这是一个表达式（即它返回一个值），句点的两边也都是表达式。这意味着可以嵌套实例变量的访问。如果 `orderTotal` 实例变量存储的是一个对象，而该对象有一个名为 `layaway` 的实例变量，则可以这样引用它：

```
boolean onLayaway = myCustomer.orderTotal.layaway;
```

句点表达式是从左向右求值的，因此首先得到的是 `myCustomer` 的 `orderTotal` 变量，它指向另一个

包含 layaway 变量的对象，因此最后得到的是变量 layaway 的值。

### 3.2.2 修改值

给变量赋值也很容易，只需在该表达式右边加上赋值运算符即可：

```
myCustomer.orderTotal.layaway = true;
```

这将变量 layaway 的值设成 true。

程序清单 3.2 中的程序检测并修改 Point 对象的实例变量。Point 位于 java.awt 包中，它指的是一个包含 x 和 y 值的坐标点。

程序清单 3.2 完整的 Pointsetter.java 源代码

---

```

1: import java.awt.Point;
2:
3: class PointSetter {
4:
5:     public static void main(String[] arguments) {
6:         Point location = new Point(4, 13);
7:
8:         System.out.println("Starting location:");
9:         System.out.println("X equals " + location.x);
10:        System.out.println("Y equals " + location.y);
11:
12:        System.out.println("\nMoving to (7, 6)");
13:        location.x = 7;
14:        location.y = 6;
15:
16:        System.out.println("\nEnding location:");
17:        System.out.println("X equals " + location.x);
18:        System.out.println("Y equals " + location.y);
19:    }
20: }
```

---

运行该应用程序时，其输出如下：

```
Starting location:
X equals 4
Y equals 13
```

```
Moving to (7, 6)
```

```
Ending location:
X equals 7
Y equals 6
```

在这个例子中，首先创建了一个 Point 实例，其中 x 等于 4，y 等于 13（第 6 行）。第 9 和 10 行使用句点表示法显示这些值。第 13 和 14 行将 x 和 y 的值分别修改为 7 和 6。最后，第 17 和 18 行再次显示 x 和 y 的值，以说明它们已被修改了。

### 3.2.3 类变量

您知道，类变量是在类中定义和存储的。它们的值适用于类及其所有实例。

每个实例都将有实例变量的一个拷贝，它们可以修改实例变量的值，而不会影响其他的实例，而类变量只有一个拷贝，修改它的值将影响所有的实例。

定义类变量的方法是，在前面加上关键字 static。例如，请看下面的类定义片段：

```

class FamilyMember {
    static String surname = "Mendoza";
    String name;
    int age;
}
```

类 `FamilyMember` 的每个实例都有自己的 `name` 和 `age` 值, 但对所有家庭成员而言, 类变量 `surname` 只有一个值: “Mendoza”。修改 `surname` 的值将影响所有的 `FamilyMember` 实例。

**注意** 之所以将类变量叫做 `static`, 是取了 `static` 的一种意思: 固定在某处。如果类有一个 `static` 变量, 则对于该类的每个对象, 该变量的值都相同。

要访问类变量, 可使用与实例变量相同的句点表示法。要取得或修改类变量的值, 可以在句点的左边使用实例名或类名。在下面范例中, 两行代码的输出相同:

```
FamilyMember dad = new FamilyMember();
System.out.println("Family's surname is: " + dad.surname);
System.out.println("Family's surname is: " + FamilyMember.surname);
```

由于可以使用实例来修改类变量的值, 因此容易对类变量及其值从何而来感到困惑。别忘了, 类变量的值将影响所有的实例。因此, 应在引用类变量时使用类名。这样, 代码将更容易阅读; 同时当出现奇怪的结果时, 调试起来也更容易。

### 3.3 调用方法

调用对象的方法与引用它的实例变量类似: 使用句点表示法。在这个方法中, 被调用的对象位于句点的左边, 方法名及其参数位于句点的右边:

```
myCustomer.addToOrder(itemNumber, price, quantity);
```

注意, 所有方法的后面都必须有圆括号, 即使该方法没有任何参数:

```
myCustomer.cancelAllOrders();
```

程序清单 3.3 的程序调用了 `String` 类中定义的一些方法。`String` 包含用于字符串检测和修改的方法, 这与其他语言的字符串库类似。

程序清单 3.3 完整的 `StringChecker.java` 源代码

```
1: class StringChecker {
2:
3:     public static void main(String[] arguments) {
4:         String str = "Nobody ever went broke by buying IBM";
5:         System.out.println("The string is: " + str);
6:         System.out.println("Length of this string: "
7:             + str.length());
8:         System.out.println("The character at position 5: "
9:             + str.charAt(5));
10:        System.out.println("The substring from 26 to 32: "
11:            + str.substring(26, 32));
12:        System.out.println("The index of the character v: "
13:            + str.indexOf('v'));
14:        System.out.println("The index of the beginning of the "
15:            + "substring \"IBM\": " + str.indexOf("IBM"));
16:        System.out.println("The string in upper case: "
17:            + str.toUpperCase());
18:    }
19: }
```

运行该程序时, 它将在系统的标准输出设备上显示下述内容:

```
The string is: Nobody ever went broke by buying IBM
Length of this string: 36
The character at position 5: y
The substring from 26 to 32: buying
The index of the character v: 8
The index of the beginning of the substring "IBM": 33
The string in upper case: NOBODY EVER WENT BROKE BY BUYING IBM
```

第 4 行使用一个字符串字面量创建了一个 `String` 实例。程序的其他部分调用了不同的字符串方法, 对该字符串进行各种操作:

- 第5行打印第4行创建的字符串的值：“Nobody ever went broke by buying IBM”。
- 第7行调用 String 对象的 length() 方法；该字符串包含 36 个字符。
- 第9行调用 charAt() 方法，该方法返回字符串中给定位置的字符。注意，字符串的起始位置为 0，而不是 1，所以位置为 5 的字符为 “y”。
- 第11行调用 substring() 方法，该方法接受两个用于指明范围的整数，并返回指定范围内的子串。调用 Substring() 方法时，也可以只提供一个参数，在这种情况下，将返回从指定位置开始到字符串末尾的子串。
- 第13行调用 indexOf() 方法，该方法返回给定字符（这里为 “v”）第一次出现的位置。字符字面量是用单引号括起的。如果第13行使用双引号将 “v” 括起，则该字面量将被视为 String。
- 第15行演示了 indexOf() 方法的另一种用法：它将一个字符串作为参数，并返回该字符串的起始位置。
- 第17行使用 toUpperCase() 方法来返回全部大写的字符串拷贝。

如果您熟悉其他编程语言中的 printf 格式，可在显示字符串时使用方法 System.out.format() 来应用这种格式。

该方法接受两个参数：输出格式以及要显示的字符串。下面的示例在给显示的整数添加美元符号和逗号：

```
int accountBalance = 5005;
System.out.format("Balance: $%,d%n", accountBalance);
```

上述代码的输出如下：

```
Balance: $5,005
```

格式字符串以百分符号 (%) 打头，后面跟一个或多个标志。格式字符串 “%,d” 显示十进制数，并将每 3 位用逗号分隔。格式字符串 “%n” 显示换行符。

下面的示例显示 pi 的值，其中包含 11 位小数：

```
double pi = Math.PI;
System.out.format("%.11f%n", pi);
```

其输出如下：

```
3.14159265359
```

#### 提示

Sun 公司的 Java 网站提供了一个有关 printf 输出的教程，该教程描述了一些最有用的格式编码，其网址为 <http://java.sun.com/docs/books/tutorial/java/data/numberformat.html>。

### 3.3.1 嵌套方法调用

方法可以返回对象的引用、基本数据类型或不返回任何值。在程序 StringChecker 中，所有调用 String 对象 str 的方法得到的返回值都被显示出来。例如，charAt() 方法返回字符串中指定位置的字符。

方法返回的值也可以被存储到变量中：

```
String label = "From";
String upper = label.toUpperCase();
```

在上面的例子中，String 对象 upper 中存储了调用 label.toUpperCase() 返回的值——文本 “FROM”，即 From 的全大写版本。

如果方法返回一个对象，则可以在同一条语句中调用该方法。这使得可以像嵌套变量那样嵌套方法。

在本章前面，介绍了一个调用时无需提供参数的方法：

```
myCustomer.cancelAllOrders();
```

如果 cancelAllOrders() 方法返回一个对象，则可以在同一条语句中调用该方法：

```
myCustomer.cancelAllOrders().talkToManager();
```

上述语句调用 `talkToManager()` 方法,它是在 `myCustomer` 的方法 `cancelAllOrders()` 返回的对象中定义的。

也可以将嵌套方法调用和实例变量引用结合起来。在下面的例子中, `putOnLayaway()` 方法是在实例变量 `orderTotal` 存储的对象中定义的,该实例变量本身位于对象 `myCustomer` 中:

```
myCustomer.orderTotal.putOnLayaway(itemNumber, price, quantity);
```

`System.out.println()` 也是一个嵌套变量和方法的例子,在所有的范例程序中,它都被用来显示信息。

`System` 类位于 `java.lang` 包中,它描述了 Java 所在系统的特有行为。`System.out` 是一个类变量,它存储了 `PrintStream` 类的一个实例。该 `PrintStream` 对象表示的是系统的标准输出,这通常是屏幕,但可被重定向到打印机或文件。对象 `PrintStream` 有一个方法 `println()`,它将一个字符串发送给输出流。

### 3.3.2 类方法

类方法与类变量一样,适用于整个类,而不是某个实例。类方法通常用作通用的工具方法,它不是直接操作某个实例而是整个类。例如,类 `String` 包含了一个名为 `valueOf()` 的类方法,它可以接受多种参数类型之一(整数、布尔型、其他对象等)。方法 `valueOf()` 返回一个 `String` 实例,其中包含参数的字符串值。这个方法并不直接操纵现有的 `String` 实例,但从另一个对象或数据类型获得一个字符串肯定是 `String` 式操作,因而在 `String` 类中定义它是合理的。

类方法还可用来将通用的方法集中起来,放在一个地方(即类中)。例如,在 `java.lang` 包中定义类 `Math`,将大量的数学运算作为类方法。类 `Math` 没有实例,但您可以使用它的类方法,并将数字和布尔值作为参数。

例如,类方法 `Math.max()` 接受两个参数,并返回其中较大的值。您不必创建 `Math` 的实例,可以在需要时随时调用它,如下所示:

```
int higherPrice = Math.max(firstPrice, secondPrice);
```

句点表示法被用来调用类方法。与类变量一样,句点的左边可以是类实例或类本身。然而,基于讨论类变量时提到的原因,使用类名将使代码更容易阅读。下述范例的最后两行的结果相同——字符串“550”:

```
String s, s2;
s = "item";
s2 = s.valueOf(550);
s2 = String.valueOf(550);
```

## 3.4 对象的引用

对处理对象而言,理解引用至关重要。

引用是一个地址,它指明了对象的变量和方法的存储位置。

将对象赋给变量或将其作为参数传递给方法时,您实际上并没有使用对象。您甚至没有使用对象的拷贝,您使用的是对象的引用。

为更好地说明这种差别,程序清单 3.4 演示了引用的工作原理。

程序清单 3.4 完整的 `RefTester.java` 源代码

```
1: import java.awt.Point;
2:
3: class RefTester {
4:     public static void main(String[] arguments) {
```

```

5:      Point pt1, pt2;
6:      pt1 = new Point(100, 100);
7:      pt2 = pt1;
8:
9:      pt1.x = 200;
10:     pt1.y = 200;
11:     System.out.println("Point1: " + pt1.x + ", " + pt1.y);
12:     System.out.println("Point2: " + pt2.x + ", " + pt2.y);
13: }
14: }

```

下面是该程序的输出：

```

Point1: 200, 200
Point2: 200, 200

```

在程序的前半部分执行如下操作。

- 第 5 行：创建两个 Point 变量。
- 第 6 行：将一个新的 Point 对象赋给 pt1。
- 第 7 行：将 pt1 的值赋给 pt2。

第 9~12 行是比较微妙的。将 pt1 的变量 x 和 y 都设置为 200，然后将 pt1 和 pt2 的所有变量都显示在屏幕上。

您可能认为 pt1 和 pt2 有不同的值。然而输出表明，情况并非如此。正如您看到的，pt2 的变量 x 和 y 也被修改了，虽然在程序中没有对它们做任何显式的修改。

这是因为第 7 行让 pt2 引用 pt1，而不是将 pt1 的拷贝赋给 pt2。

pt2 引用的对象与 pt1 相同，如图 3.1 所示。它们都可用来引用该对象或修改它的变量。

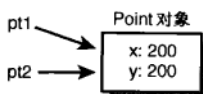


图 3.1 对象的引用

要让 pt1 和 pt2 指向不同的对象，可以在第 6 行和第 7 行分别使用 new Point() 语句来创建不同的对象，如下所示：

```

pt1 = new Point(100, 100);
pt2 = new Point(100, 100);

```

在 Java 中，将参数传递给方法时，引用变得相当重要。这将在本章后面做更详细的介绍。

#### 注意

在 Java 中，没有 C 和 C++ 中那样的显式指针和指针算术。然而，使用引用和 Java 数组，可以实现大多数指针功能，但不会有其中的许多缺点。

### 3.5 对象和基本数据类型的转换和强制类型转换

Java 对其处理的信息非常苛刻。像 Morris (9Lives 猫食广告中永不满足的猫) 一样，Java 希望所有的事情都是确定的，它不能容忍任何可选的情况出现。

将参数传递给方法或在表达式中使用变量时，必须使用数据类型正确的变量。如果方法要的是 int，而您试图传递一个 float 值，Java 编译器将报错。同样，使用另一个变量的值来设置某个变量时，它们的类型必须相同。

有时，Java 程序中某个值的类型并非您所需要的。这可能是由于类或数据类型不合适，例如您需要 int，而它是一个 float。

**注意**

Java 编译器有一个方面不像 Morris 那样苛刻: String 拼接运算符(+)简化了方法 println()、赋值语句和方法参数中的字符串操作。如果被拼接的变量中有一个是字符串,则 Java 将所有的内容都作为字符串进行处理。这使得下面的操作成为可能:

```
float gpa = 2.25F;  
System.out.println("Honest, dad, my GPA is a " + (gpa+1.5));
```

在 Java 中,通过使用拼接运算符,可以用一个字符串存储多个对象和基本数据类型变量的文本表示。

可以通过强制类型转换将值从一种类型转换为另一种类型。

强制类型转换 (casting) 生成一个类型不同于源值的新值。

虽然强制类型转换的概念很简单,但使用起来却很复杂,这是由于 Java 既有基本数据类型 (如 int、float 和 boolean),又有对象类型 (String、Point、ZipFile 等)。本节将介绍 3 种形式的强制类型转换:

- 基本类型之间的强制类型转换,如 int 到 float 或 float 到 double;
- 从一种类型的实例强制转换为另一种类型的实例,如 Object 到 String;
- 从基本数据类型强制转换为对象,然后从对象中提取出基本类型值。

讨论强制类型转换时,以源和目标的方式考虑问题将更容易。源是要被强制转换为另一种类型的变量,目标是转换后的结果。

### 3.5.1 强制转换基本类型

基本类型之间的强制转换让您能够将值从一种基本数据类型转换为另一种。这通常发生在数字类型上,而有一种基本数据类型永远不能用于强制类型转换中。布尔值要么为 true,要么为 false,不能用于强制类型转换操作中。

在许多基本类型间的强制转换中,目标可以保存比源更大的值,因此转换起来很容易。例如,将 byte 转换为 int。因为 byte 的取值范围为-128~127,而 int 的取值范围为-2100000~21000000,所以将 byte 转换为 int 时,有足够多的空间来存储。

通常可以将 byte 或 char 用作是 int、将 int 用作 long 或 float,将任何数字类型用作 double。在大多数情况下,由于更大的数据类型的精确度比小类型高,所以不会导致信息丢失。一种例外情况是将整数转换为浮点数。将 int 或 long 转换为 float 或将 long 转换为 double 时,都可能导致精确度降低。

**注意**

字符可以被用作 int,因为每个字符都有相应的数字编码,它表示该字符在字符集中的位置。如果变量 i 的值为 65,则强制类型转换 (char) i 的结果为字符 A。在 ASCII 字符集中, A 对应的数字编码是 65。这种字符集是 Java 支持的字符的一部分。

将大类型值转换为小类型值时,必须显式地进行强制类型转换,否则将导致精确度降低。显式强制类型转换的格式如下:

```
(typename) value
```

在上面的例子中,typename 是目标数据类型,如 short、int 或 float。value 是表达式,其结果为要转换的源类型。例如,在下面的例子中,x 被 y 除后的结果被强制转换为 int 类型:

```
int result = (int)(x / y);
```

注意,由于强制类型转换的优先级高于算术运算,所以这里必须使用圆括号,否则,x 将被首先转换为 int,然后再被 y 除,从而得出错误的结果。



### 3.5.2 强制转换对象

类的实例也可被转换为其他类的实例，但有限制：源和目标类必须有继承关系，即其中一个必须是另一个的子类。

与将基本类型转换为更大的类型相似，有些对象无需被显式地转换。具体地说，由于子类包含了超类的所有信息，因此可以在任何期望超类的地方使用子类的实例。

例如，来看一个方法，它接受两个参数：一个为 `Object` 类型，另一个为 `Component` 类型。可以将任何类的实例作为 `Object` 参数，因为所有 Java 类都是 `Object` 的子类。对 `Component` 参数，可以给它传递其子类，如 `Button`、`Container` 和 `Label`。

在程序的任何地方都是如此，而不仅仅是在方法调用中。如果有一个 `Component` 变量，则可以将该类或其子类的任何对象赋给该变量，而无需进行强制类型转换。

反过来也如此，可以在需要子类的地方使用超类。然而这有一个问题：由于子类的行为比超类多，所以将降低精度。超类对象可能不具备子类对象的所有行为。例如，如果在调用 `Integer` 对象的方法的地方使用 `Number` 对象，将不包括许多 `Integer` 特有的方法。试图调用目标对象没有的方法时将出错。

要在需要子类对象的地方使用超类对象，必须显式地进行强制类型转换。在转换过程中，不会损失任何信息，而是得到了子类定义的全部方法和变量。要将对象强制转换为另一种类，需要使用与基本类型相同的操作：

```
(classname)object
```

其中，`classname` 是目标类的名称，`object` 是源对象的引用。注意，强制类型转换将创建一个 `classname` 对象的引用，原来的对象继续存在。

下面的范例将类 `VicePresident` 的一个实例强制转换为类 `Employee` 的实例。`VicePresident` 是 `Employee` 的子类，其中包含更多信息：

```
Employee emp = new Employee();
VicePresident veep = new VicePresident();
emp = veep; // no cast needed for upward use
veep = (VicePresident)emp; // must cast explicitly
```

使用 `Java2D` 图形操作时，必须强制转换对象的类型。您必须将 `Graphics` 对象强制转换为 `Graphics2D` 对象后，才能在屏幕上画图或文本。下面的例子使用了一个名为 `screen` 的 `Graphics` 对象来新建一个名为 `screen2D` 的 `Graphics2D` 对象。

```
Graphics2D screen2D = (Graphics2D)screen;
```

`Graphics2D` 是 `Graphics` 的子类，这两种类都位于 `java.awt` 包中。第 13 章将全面地介绍这一主题。

除了强制转换为某种类之外，还可以将对象强制转换为接口，但仅当该对象的类或其超类之一实现了该接口时才行。将对象强制转换为接口意味着您可以调用该接口的方法，即使该对象的类并没有实现这个接口。

### 3.5.3 基本类型和对象之间的转换

在任何情况下，您都不能将对象强制转换为基本数据类型或将基本类型强制转换为对象。在 Java 中，基本类型和对象是完全不同的东西，不能自动在两者之间强制转换或互换。

在 Java 中，基本类型和对象是完全不同的东西，不会自动在它们之间进行转换。

`java.lang` 包中包含了对应于每种基本数据类型的类：`Float`、`Boolean`、`Byte` 等。这些类大多数与数据类型的名称相同，只是类名首字母大写（即 `Short`，而不是 `short`；`Double`，而不是 `double`）。

等)。另外，有两个类的名称与对应数据类型不同，即分别对应于 `char` 和 `int` 变量的 `Character` 和 `Integer`。

使用每个基本类型对应的类，可以创建存储相同值的对象。下面的语句创建了类 `Integer` 的一个实例，其值为 7801：

```
Integer dataCount = new Integer(7801);
```

使用这种方法创建了对象后，可像使用其他对象那样使用它（虽然不能修改它的值）。当您想将它作为基本类型值使用时，也有用于实现这种方法。例如，要从对象 `dataCount` 获得一个 `int` 值，可以使用下面的语句：

```
int newCount = dataCount.intValue(); // returns 7801
```

在程序中，常常需要将字符串转换为数字类型，如整数类型。需要 `int` 结果时，可以使用 `Integer` 类的类方法 `parseInt()` 来实现，如下例所示：

```
String pennsylvania = "65000";
int penn = Integer.parseInt(pennsylvania);
```

下面的类用来处理对象，而不是基本数据类型：`Boolean`、`Byte`、`Character`、`Double`、`Float`、`Integer`、`Long`、`Short` 和 `Void`。这些类通常被称为对象封装器（object wrapper），因为它们提供了基本类型值的对象表示。

#### 警告

如果您在程序中使用上述代码，该程序将无法通过编译。如果参数不是合法的数值，`parseInt()` 将发生 `NumberFormatException` 错误。为处理这种错误，必须使用特殊的错误处理语句，这将在第 7 章中介绍。

通过自动封装（autoboxing）和拆封（unboxing）——一种自动转换过程，处理表示同一类值的基本类型和对象时更为容易。

自动封装自动地将基本类型转换为对象，而拆封执行相反的操作。

如果编写语句时，在本应使用基本类型变量的地方使用了一个对象或相反，相应的值将被转换，以便语句能够被成功地执行。

这是与以前的 Java 版本显著不同的地方。

例如，根据 Java 2 1.4 版的规则，下述语句将不能通过编译：

```
Float f1 = new Float(12.5F);
Float f2 = new Float(27.2F);
System.out.println("Lower number: " + Math.min(f1, f2));
```

当您试图编译包含上述语句的类时，编译器将停止编译，并显示一条错误消息，指出方法 `Math.min()` 接受两个 `float` 参数，而不是 `Float` 对象。

在 Java 6 中，上述语句将通过编译。当方法 `Match.min()` 被调用时，`Float` 对象将被自动拆封为相应的基本类型值。

#### 警告

仅当对象包含值时才能对它进行拆封。如果没有调用构造函数来创建对象，编译将失败：发生 `NullPointerException` 错误。

## 3.6 比较对象值和类

除强制类型转换外，还常常需要对对象执行下列 3 种操作：

- 比较对象；
- 判断对象所属的类；
- 判断对象是否是特定类的实例。

### 3.6.1 比较对象

前一章介绍了用于对值进行比较的运算符：等于、不等于、小于等。这些运算符中的大部分都只能用于基本类型，而不能用于对象。如果将非基本类型值作为操作数，Java 编译器将报错。

对于这一规则，一种例外情况是用于相等关系的运算符：`==`（等于）和`!=`（不等）。用于对象时，这些运算符的功能并非您期望的那样。它们不是检查一个对象的值是否与另一个对象相同，而是判断运算符两边引用的是否是同一个对象。

要比较类实例，并使结果有意义，必须在类中实现特殊的方法，并调用这些方法。

一个很好的例子是类 `String`。两个不同的 `String` 对象可能包含相同的值。然而，如果使用 `==` 运算符来比较它们，则它们将被认为不相等。虽然它们的内容一致，但它们不是同一个对象。

要检查两个 `String` 对象的值是否相同，可使用其 `equals()` 方法。该方法检测字符串中的每个字符，如果两个字符串的值相同，则返回 `true`。程序清单 3.5 演示了这一点。

程序清单 3.5 完整的 `EqualsTester.java` 源代码

```
1: class EqualsTester {
2:     public static void main(String[] arguments) {
3:         String str1, str2;
4:         str1 = "Free the bound periodicals.";
5:         str2 = str1;
6:
7:         System.out.println("String1: " + str1);
8:         System.out.println("String2: " + str2);
9:         System.out.println("Same object? " + (str1 == str2));
10:
11:        str2 = new String(str1);
12:
13:        System.out.println("String1: " + str1);
14:        System.out.println("String2: " + str2);
15:        System.out.println("Same object? " + (str1 == str2));
16:        System.out.println("Same value? " + str1.equals(str2));
17:    }
18: }
```

该程序的输出如下：

```
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? false
Same value? true
```

程序的前半部分（第 3~5 行）声明了两个变量（`str1` 和 `str2`），将字面量“Free the bound periodicals.”赋给 `str1`，然后将 `str1` 赋给 `str2`。正如前面介绍的，现在 `str1` 和 `str2` 指向同一个对象，第 9 行的相等性测试证明了这一点。

该程序的后半部分创建一个新的 `String` 对象（其值与 `str1` 相同），并赋给它 `str2`。现在，`str1` 和 `str2` 指向两个不同的字符串对象，但它们的内容相同。使用 `==` 运算符（第 15 行）来检测它们是否是同一个对象时，得到预料之中的答案（`false`，它们在内存中不是同一个对象）。第 16 行使用 `equals()` 方法来检测它们，得到的结果也是预料之中的（`true`，它们的值相同）。

#### 注意

为什么修改 `str2` 时，要使用 `new` 而不使用另一个字面量？字符串字面量在 Java 中是经过优化的：如果使用字面量创建一个字符串，然后用相同的字符内容来重新创建它时，Java 将返回原来的 `String` 对象。这样，两个字符串将是同一个对象——您并没有创建两个不同的对象。

### 3.6.2 判断对象所属的类

想确定对象所属的类，可以采用下述方式。这里判断的是赋给变量 `key` 的对象所属的类：

```
String name = key.getClass().getName();
```

该语句有何功能呢？方法 `getClass()` 是在 `Object` 类中定义的，因此所有的对象都有该方法。该方法返回的是一个 `Class` 对象（`Class` 是一种类），对象的 `getName()` 返回一个表示类名的字符串。

另一种检测方式是 `instanceof` 运算符。`Instanceof` 使用两个操作数：左边为对象的引用，右边是类名。该表达式返回一个布尔值：如果该对象是这种类或其子类的实例，为 `true`，否则为 `false`，如下面的例子所示：

```
boolean check1 = "Texas" instanceof String // true
```

```
Point pt = new Point(10, 10);
boolean check2 = pt instanceof String // false
```

`instanceof` 运算符还可用于接口。如果对象实现了某个接口，则使用运算符 `instanceof` 测试该接口时，结果将为 `true`。

## 3.7 总结

至此，您已通过三章学习了 Java 是如何实现面向对象编程的，可以更准确地判断它在编程中的用途。

如果您是悲观主义者，面向对象编程将更加抽象，它妨碍您使用编程语言来完成工作。接下来的几章，将更深入地说明为何 OOP 是 Java 不可分割的部分。

如果您是乐观主义者，面向对象编程技术将是值得使用的，因为它有很多优点：提高了可靠性、可重用性和可维护性。

在本章中，您学习了如何处理对象，即创建对象、读取和修改它们的值、调用它们的方法。您还学习了如何将对象从一种类强制转换为另一种类、从一种数据类型强制转换为一种类以及使用自动封装和拆封功能实现自动转换。

## 3.8 问与答

问：我不太明白对象与基本数据类型（如 `int` 和 `boolean`）之间的差别。

答：基本数据类型（`byte`、`short`、`int`、`long`、`float`、`double`、`boolean` 和 `char`）表示语言中最小的元素。它们不是对象，虽然在很多情况下处理起来与对象类似——可以被赋给变量，被传递到方法或从方法返回。

对象是类的实例，因此通常是比数字和字符复杂得多的数据类型，并常常将数字和字符作为实例变量和类变量。

问：程序清单 3.3 中的 `length()` 和 `charAt()` 方法看上去有些不太合理。如果 `length()` 表明，字符串包含 36 个字符，则使用 `charAt()` 来显示字符串中的字符时，不也应使用 1~36 的数字进行编号吗？

答：这两个方法看待字符串时有些差别。方法 `length()` 计算字符串中的字符个数，第一个字符计为 1，第二个为 2，以此类推。字符串“Charlie Brown”有 13 个字符。方法 `charAt()` 认为字符串中第一个字符的编号为 0，这与 Java 中数组元素的编号相同。字符串 Charlie Brown 中的字符编号为 0~12，即字母 C 到字母 n。

问：如果 Java 没有指针，如何进行类似链表的操作呢？在链表中，使用指针来从一个节点链接到另一个节点，从而可以在它们之间移动。

答：不能说 Java 没有指针，它只是没有显式指针。对象引用实际上就是指针。要创建类似链表的结构，可以创建一个 Node 类，其中包含一个 Node 类型的实例变量。要将节点对象连接起来，可将节点对象赋给它前面一个对象的实例变量。因为对象引用是指针，这样创建链表将具备您期望的行为（在第 8 章中，将使用 Java 类库中的链表）。

## 3.9 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 3.9.1 问题

1. 哪个运算符可以用来调用对象的构造方法，以创建新的对象？
  - a. +;
  - b. new;
  - c. instanceof.
2. 哪种方法适用于类的所有对象，而不是某个对象？
  - a. 通用方法；
  - b. 实例方法；
  - c. 类方法。
3. 如果程序中包含名为 obj1 和 obj2 的对象，则使用语句 obj2=obj1 时将发生什么情况？
  - a. obj2 的实例变量的值将与 obj1 相同；
  - b. obj2 和 obj1 是同一个对象；
  - c. a 和 b 都不对。

答案

1. b。
2. c。
3. b，运算符=并不将一个对象的值复制到另一个对象中。相反，它将两个变量指向同一个对象。

### 3.9.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

在下面的代码中：

```
public class AyeAye {
    int i = 40;
    int j;

    public AyeAye() {
        setValue(i++);
    }

    void setValue(int inputValue) {
        int i = 20;
        j = i + 1;
        System.out.println("j = " + j);
    }
}
```



```
}  
}
```

当 setValue ( ) 方法显示变量 j 时, 该变量的值为多少?

- a. 42,
- b. 40,
- c. 21,
- d. 20。

答案 c

### 3.10 练习

为巩固本章介绍的知识, 请尝试完成下面的练习。

1. 创建一个程序, 它将 MM/DD/YYYY 格式 (如 12/04/2007) 的生日转换为 3 个不同的字符串。
2. 创建一个类, 它含有实例变量 height、weight 和 depth, 这些变量的类型都为 int。创建一个 Java 程序, 它使用这个新类, 并给对象的这些实例变量赋值, 然后显示这些值。



## 第 4 章

# 数组、逻辑和循环

本章介绍 Java 语言中 3 种最烦人的特性：

- 使用循环来重复执行 Java 程序中的部分代码；
- 让程序根据某种逻辑来决定是否执行某项操作；
- 使用数组将类或数据类型相同的数据进行分组。

如果您认为这些特性不烦人，那就不烦人。您使用 Java 完成的大部分重要的工作都将大量地使用这 3 种特性。

对计算机来说，这些特性有些烦。因为它们让软件完成其最擅长的工作之一：负责完成重复的任务。

### 4.1 数 组

至此，您在每个 Java 程序中只处理几个变量。在某些情况下，使用单个变量来存储信息是可行的，但如果需要记录 20 条相关的信息，该如何办呢？可以创建 20 个不同的变量，并给它们指定初值，但随着处理的信息量越来越大，工作也将越来越烦琐。如果有 100 项甚至 1000 项，该如何办呢？

数组用来存储一系列数据项，其中的每一项具有相同的基本数据类型、类或相同的父类。每一项都有自己的位置，因此访问起来很容易。

数组可能的类型与变量相同，但数组被创建后，就只能用来存储指定类型的信息。例如，可以有 `int` 数组、`String` 对象的数组或数组的数组，但同一个数组不能同时存储 `String` 对象和 `int` 值。

Java 实现数组的方式与其他编程语言不同——因为对象可以被当作其他对象进行处理。

要创建数组，必须这样做：

1. 声明一个用于存储数组的变量；
2. 创建一个数组对象，并将它赋给数组变量；
3. 在数组中存储信息。

#### 4.1.1 声明数组变量

创建数组时，首先需要声明一个用来存储数组的变量。数组变量指出了数组将存储的对象或数据类型以及数组的名称。为将其与常规变量声明区别开来，需要将一对方括号 (`[]`) 添加到对象、数据类型或变量名后面。

下面的语句是数组变量声明的例子：

```
String[] requests;  
Point[] targets;  
float[] donations;
```

声明数组时，还可以将方括号放在变量名，而不是类型后面，如下所示：

```
String requests[];

Point targets[];

float donations[];
```

#### 注意

使用哪种格式取决于个人喜好。本书的范例程序都将方括号放在信息类型后，而不是变量名后。在 Java 程序员中，这种方法更为流行。

### 4.1.2 创建数组对象

声明数组变量后，接下来需要创建一个数组对象，并将它赋给数组变量。为此需要：

- 使用 new 运算符；
- 直接初始化数组的内容。

由于在 Java 中，数组是对象，因此可以使用 new 运算符来创建新的数组实例，如下所示：

```
String[] players = new String[10];
```

该语句创建一个新的字符串数组，它包含 10 个可用于存储 String 对象的元素。使用 new 来创建数组对象时，必须指定数组的大小。上述语句并没有将 String 对象存储到元素中，您必须在后面完成这项任务。

就像可以存储对象一样，数组对象也可以存储诸如 int 或 Boolean 等基本数据类型：

```
int[] temps = new int[99];
```

使用 new 创建数组对象时，其所有元素都被自动地初始化（数字数组为 0，布尔数组为 false，字符数组为 '\0'，对象数组为 null）。

#### 注意

Java 关键字 null 指的是 null 对象（可用于任何对象引用）。它并不像在 C 语言中的 NULL 常量那样，为零或字符 '\0'。

由于对象数组被创建时，数组中每个对象的引用皆为 null，因此使用之前必须给将对象赋给每个数组元素。

下面的例子创建 1 个由 3 个 Integer 对象组成的数组，然后将对象赋值给每个元素：

```
Integer[] series = new Integer[3];
series[0] = new Integer(10);
series[1] = new Integer(3);
series[2] = new Integer(5);
```

也可以在创建数组的同时，对其进行初始化，方法是将数组元素放在大括号中，并用逗号分隔：

```
Point[] markup = { new Point(1,5), new Point(3,3), new Point(2,3) };
```

大括号中的每个元素的数据类型都必须与数组的数据类型相同。采用这种方式来创建并初始化数组时，数组的大小与大括号中包含的元素数目相同。前面的例子创建了一个名为 markup 的 Point 对象数组，它包含了 3 个元素。

由于 String 对象可以不用 new 运算符来创建并初始化，因此创建字符串数组时，可以这样进行初始化：

```
String[] titles = { "Mr.", "Mrs.", "Ms.", "Miss", "Dr." };
```

上面的语句中创建了一个名为 titles 的 String 对象数组，该数组包含 5 个元素。

所有数组都有一个名为 length 的实例变量，它指出了数组中包含多少个元素。例如，titles.length 的值为 4。



### 4.1.3 访问数组元素

创建并初始化数组后，便可以读取、修改和检查数组中各个元素的值。访问元素值的方法是使用数组名和用方括号括起下标。名称和下标可用于表达式中，如下所示：

```
testScore[40] = 920;
```

上面的语句将数组 `testScore` 的第 41 个元素的值设置为 920。其中的 `testScore` 是存储数组的变量，虽然这也可以是一个结果为数组的表达式。下标表达式指出要访问哪个元素。

数组第一个元素的下标为 0，而不是 1，因此在包含 12 个元素的数组中，各个元素的下标分别为 0~11。

数组下标将被检查，以确保它位于数组边界之内，该边界是在创建数组时指定的。在 Java 中，要对超出数组边界的元素进行访问或赋值是不可能的，这就避免了 C 语言中由于数组越界所引发的问题。请看下面的两条语句：

```
float[] rating = new float[20];
rating[20] = 3.22F;
```

当包含上面两行语句的程序用到 `rating[20]` 时，将产生编译错误。这是因为数组 `rating` 没有编号为 20 的元素——它总共有 20 个元素，编号为 0~19。Java 编译器将显示错误消息“`ArrayIndexOutOfBoundsException`”，以指出这一点。

如果数组下标是在程序运行时计算得到的，而它超出了数组的边界，则 Java 解释器也将产生错误。解释器将产生一个异常，以指出这种错误。有关异常以及如何使用它，将在第 7 章做更详细的介绍。

避免在程序中无意间超越数组边界的方法之一是，使用 `length` 实例变量，所有数组对象都有这样的变量，不管数组类型是什么。`Length` 变量包含了数组中的元素个数。下面的语句显示 `rating` 对象中的元素个数：

```
System.out.println("Elements: " + rating.length);
```

### 4.1.4 修改数组元素

正如前面的范例所示，可以给数组中的元素赋值，方法是在数组名和下标后面加上赋值运算符和要指定的值，如下所示：

```
temperature[4] = 85;
```

```
day[0] = "Sunday";
```

```
manager[2] = manager[0];
```

需要指出的一点是，Java 中的对象数组是一组到对象的引用。将对象赋给这种数组中的元素时，将创建一个到该对象的引用。移动数组中的值，是在重新指定引用，而不是将值从一个元素复制到另一个中。对于基本数据类型的数组（如 `int` 或 `float`），这种操作实际上是将值从一个元素复制到另一个元素中；`String` 数组也是如此，虽然它们是对对象。

数组的创建和修改都相当简单，但它们为 Java 提供了为数众多的功能。程序清单 4.1 是一个简单的程序，它创建并初始化 3 个数组，然后显示这些数组的元素。

程序清单 4.1 完整的 `HalfDollars.java` 源代码

```
1: class HalfDollars {
2:     public static void main(String[] arguments) {
3:         int[] denver = { 2500000, 2900000, 3500000 };
4:         int[] philadelphia = new int[denver.length];
5:         int[] total = new int[denver.length];
6:         int average;
7:     }
```

```

8:      philadelphia[0] = 2500000;
9:      philadelphia[1] = 2900000;
10:     philadelphia[2] = 3800000;
11:
12:     total[0] = denver[0] + philadelphia[0];
13:     total[1] = denver[1] + philadelphia[1];
14:     total[2] = denver[2] + philadelphia[2];
15:     average = (total[0] + total[1] + total[2]) / 3;
16:
17:     System.out.print("2003 production: ");
18:     System.out.format("%,d\n", total[0]);
19:     System.out.print("2004 production: ");
20:     System.out.format("%,d\n", total[1]);
21:     System.out.print("2005 production: ");
22:     System.out.format("%,d\n", total[2]);
23:     System.out.print("Average production: ");
24:     System.out.format("%,d\n", average);
25: }
26: }

```

应用程序 HalfDollars 使用 3 个 int 数组来存储 Denver 和 Philadelphia 造币厂生产的 50 美分硬币的总量。该程序的输出如下：

```

2003 production: 5,000,000
2004 production: 5,800,000
2005 production: 7,300,000
Average production: 6,033,333

```

这里创建的类 HalfDollars 包含 3 个实例变量，这些变量存储 int 数组。

第一个名为 denver，是在第 3 行声明并初始化的，它包含了 3 个整数：元素 0 的值为 2500000，元素 1 为 2900000，元素 2 为 3500000。这些数字是 Denver 造币厂 3 年生产的 50 美分硬币的总量。

第 2 和第 3 个实例变量 (philadelphia 和 total) 是在第 4、5 行声明的。数组 philadelphia 包含了 Philadelphia 造币厂的生产总量，total 用来存储两个地方合计的生产总量。

在第 4 和第 5 行没有为数组 philadelphia 和 total 的元素赋初值。因此，每个元素的值都为默认值 0。

变量 denver.length 用来将这两个数组的元素数目指定为与数组 denver 相同。每个数组都有 length 变量，使用它可以知道数组包含的元素数目。

在这个应用程序中，main() 方法中的其他代码执行如下操作。

- 第 6 行创建一个名为 average 的 int 变量。
- 第 8~10 行将新值赋给数组 philadelphia 的 3 个元素。
- 第 12~14 行给数组 total 的元素赋值。在第 12 行中，将 denver 的元素 0 与 philadelphia 的元素 0 之和赋给 total 的元素 0。第 13 和 14 行使用了类似的表达式。
- 第 15 行将变量 average 的值设置为 3 个 total 元素的平均值。由于 total 的 3 个元素以及 average 都是整数，因此平均值也为整数，而不是浮点数。
- 第 17~24 行显示存储在数组 total 和变量 average 中的值。这里使用了方法 System.out.format()，通过添加逗号使显示的数字更容易阅读。

该应用程序以一种低效的方式处理数组。除用来指明要引用哪个数组元素的下标不同外，这些语句几乎相同。如果应用程序 HalfDollars 用来跟踪 100 年的总产量，而不是 3 年的产量，该程序将包含大量重复的代码。

操纵数组时，可以使用循环来遍历数组元素，而不是分别进行处理。这样，代码将更加简短，也更容易阅读。在本章后面介绍循环后，将重新编写该程序。

### 4.1.5 多维数组

如果您使用过其他语言中的数组，则可能认为 Java 也将支持多维数组——这种数组包含多个下标，

可以以多维的方式存储信息。

多维数组经常用于表示在  $x, y$  格点数组元素中的数据。

Java 不支持多维数组,但可以通过声明数组的数组来实现同样的功能。这些数组还可以包含数组,依次类推,最后达到所需维数。

例如,假设一个程序需要完成以下任务:

- 对一年中的每一天,都记录一个整数;
- 将这些值分周进行组织。

为组织这些数据,一种方法是创建一个包含 52 个元素的数组,其中的每个元素又是一个包含 7 个元素的数组:

```
int[][] dayValue = new int[52][7];
```

该数组的数组总共包含 364 个整数,每天一个。可以使用下面的语句来设置第 10 周的第一天的值:

```
dayValue[9][0] = 14200;
```

也可以使用这些数组的实例变量 *length*,就像其他数组一样。下面的语句声明了一个三维 *int* 数组,并显示了每一维的元素数:

```
int[][][] century = new int[100][52][7];
System.out.println("Elements in the first dimension: " + century.length);
System.out.println("Elements in the second dimension: " + century[0].length);
System.out.println("Elements in the third dimension: " + century[0][0].length);
```

## 4.2 块语句

Java 中的语句被组织为块。块以花括号开始和结束——左花括号 ({) 表示开始,右花括号 (}) 表示结束。

前 4 章的程序中都使用了花括号。它们被用来在类定义中包含变量和方法以及定义属于某个方法的语句。

块也叫做块语句 (block statements),因为整个块可用在任何可使用单条语句的地方(在 C 和其他语言中,它们被称为复合语句)。块中语句从上到下依次被执行。

块可以放在其他块中,就像将方法放在类定义中一样。

使用块时,需要注意的重要的一点是,它为块中声明的局部变量创建了作用域。

作用域是程序的一部分,在其中变量存在并可使用。如果在变量的作用域外要使用它,将发生错误。

在 Java 中,变量的作用域声明它的语句所在的块。可以在块中声明和使用局部变量,在该块执行完毕后,这些变量将不复存在。例如,下面的 *testBlock()* 方法包含一个块:

```
void testBlock() {
    int x = 10;
    { // start of block
        int y = 40;
        y = y + x;
    } // end of block
}
```

在这个方法中,定义了两个变量:  $x$  和  $y$ 。变量  $y$  的作用域是它所在的块,因此只能在该块内被使用。试图在方法 *testBlock()* 的其他部分使用变量  $y$  将出错。变量  $x$  是在方法内(但在内层块之外)创建的,因此可用于方法的任何地方。可以在方法内的任何地方修改  $x$  的值,而且该值将保留下来。

块语句并不仅仅用于方法定义中,可以在类定义和方法定义中使用它们,也可以在接下来将要介绍的逻辑和循环结构中使用它们。

## 4.3 if 条件语句

编程的一个关键方面在于,程序能够判断它应该做什么。这是通过条件语句来实现的。

条件语句是一种编程语句，仅当指定的条件满足时它才被执行。

最基本的条件语句是 if 关键字。if 条件语句使用布尔表达式来判断是否执行语句。如果表达式返回 true，则执行语句。

下面是一个简单的例子，它根据一个条件来决定是否显示消息 “Not enough arguments”：实例变量 arguments.length 的值是否小于 3：

```
if (arguments.length < 3)
    System.out.println("Not enough arguments");
```

如果想在 if 表达式返回 false 值时执行其他操作，可使用可选关键字 else。下面的例子使用了 if 和 else：

```
int duration;
if (arguments.length < 1)
    ""server = "localhost";
else
    ""server = arguments[0];
```

if 条件语句根据单个布尔测试的结果执行不同的语句。

#### 注意

在 Java 中的 if 语句与在其他语言中的区别在于，Java 要求测试返回布尔值(true 或 false)。而在 C 语言中，测试可以返回整数值。

使用 if 语句时，只能将一条语句作为测试表达式为真才执行的代码，将另一条语句作为测试为假才执行的代码。

然而，正如本章前面指出的，在 Java 中，块可以出现在任何可以使用单条语句的地方。要在 if 语句中完成多项操作，而不仅仅是一项，可以将这些语句放在块中。请看下面的程序片段，它摘自第 2 章：

```
if (temperature > 660) {
    status = "returning home";
    speed = 5;
}
```

其中 if 语句包含了测试表达式 temperature>660。如果变量 temperature 的值大于 660，块语句将被执行，因而发生如下两件事：

- 将变量 status 的值设置为 returning home；
- 将变量 speed 的值设置为 5。

如果 temperature 小于或等于 660，整个语句块都将被跳过，而不会发生任何情况。

所有 if 和 else 语句都使用布尔测试来判断是否执行语句。可将布尔变量用作测试表达式，如下所示：

```
if (outOfGas)
    status = "inactive";
```

上面的例子使用了一个名为 outOfGas 的布尔变量，其功能与下面的代码等效：

```
if (outOfGas == true)
    status = "inactive";
```

## 4.4 switch 条件语句

在任何语言中，都常常需要将变量同某个值进行比较，如果不匹配，再同另一个值来进行比较，依次类推。如果使用 if 语句，这可能很烦琐，这取决于需要同多少个值进行比较。例如，最后的代码可能如下：

```
if (operation == '+')
    add(object1, object2);
else if (operation == '-')
    subtract(object1, object2);
else if (operation == '*')
    multiply(object1, object2);
else if (operation == '/')
    divide(object1, object2);
```

if 语句的这种用法叫做嵌套 if 语句，因为每个 else 语句又包含一个 if，直到所有可能的测试全部完成。

在有些编程语言中，一种简化嵌套 if 语句的机制是，将测试和操作组合在一起，放到一条语句中。在 Java 中，可以使用 switch 语句来将操作组织起来，这与 C 语言的情况相同。下面是一个使用 switch 语句的例子：

```
switch (grade) {
    case 'A':
        System.out.println("Great job!");
        break;
    case 'B':
        System.out.println("Good job!");
        break;
    case 'C':
        System.out.println("You can do better!");
        break;
    default:
        System.out.println("Consider cheating!");
}
```

switch 语句基于测试。在上面的例子中，测试的是变量 grade 的值，该变量存储的是一个 char 值。测试变量可以是任何基本数据类型，如 byte、char、short 或 int，它将依次与 case 中的每个值进行比较。如果找到匹配的情况，则执行相应的语句。

如果没有找到匹配的值，则执行 default 语句。default 语句是可选的，如果被省略，则当没有任何 case 匹配时，将不执行任何操作。

Java 中的 switch 实现有一定的限制：测试和值都只能是可被转换为 int 的基本数据类型。不能在 switch 中使用更大的数据类型，如 long、float、字符串或其他对象，也不能测试除相等性以外的任何关系。这些限制使得 switch 只能用于最简单的情况。而嵌套的 if 语句可以用于任何类型的任何测试。

下面是前面嵌套 if 语句的修订版本，它使用的是 switch 语句：

```
switch (operation) {
    case '+':
        add(object1, object2);
        break;
    case '-':
        subtract(object1, object2);
        break;
    case '*':
        multiply(object1, object2);
        break;
    case '/':
        divide(object1, object2);
        break;
}
```

在每个 case 后，可以有一条或多条语句——事实上，可以有任何数目的语句。与 if 语句不同，不必将多条语句用花括号括起。

每个 case 中都有一个 break 语句，用于指出何时停止执行语句。如果 case 中没有 break 语句，则找到匹配的情况后，该 case 中的语句及其后到 break 或 switch 末尾之前的所有语句都将执行。

在某些情况下，这确实是您想要进行的操作；然而，在大多数情况下，应该包含 break 语句，以确保只执行相应的代码。在 4.7 节中，您将知道，break 语句跳到下一个闭括号后面的代码处。

一种不需要 break 的情况是，对于多个不同的值，都执行相同的语句。为此，可以使用多个 case 行，switch 将执行它找到第一条语句。

例如，在下面的 switch 语句中，如果 x 的值为 2、4、6 或 8，将打印字符串“x is an even number.”；在其他情况下，将打印“x is an odd number.”。

```
switch (x) {
    case 2:
    case 4:
    case 6:
    case 8:
```

```
        case 8:
            System.out.println("x is an even number");
            break;
        default: System.out.println("x is an odd number");
    }
}
```

在程序清单 4.2 中, 应用程序 DayCounter 接受 2 个参数: 月份和年份, 并显示指定月份的天数。这里使用了 switch 语句、if 语句和 else 语句。

程序清单 4.2 完整的 DayCounter.java 源代码

```
1: class DayCounter {
2:     public static void main(String[] arguments) {
3:         int yearIn = 2008;
4:         int monthIn = 1;
5:         if (arguments.length > 0)
6:             monthIn = Integer.parseInt(arguments[0]);
7:         if (arguments.length > 1)
8:             yearIn = Integer.parseInt(arguments[1]);
9:         System.out.println(monthIn + "/" + yearIn + " has "
10:            + countDays(monthIn, yearIn) + " days.");
11:     }
12:
13:     static int countDays(int month, int year) {
14:         int count = -1;
15:         switch (month) {
16:             case 1:
17:             case 3:
18:             case 5:
19:             case 7:
20:             case 8:
21:             case 10:
22:             case 12:
23:                 count = 31;
24:                 break;
25:             case 4:
26:             case 6:
27:             case 9:
28:             case 11:
29:                 count = 30;
30:                 break;
31:             case 2:
32:                 if (year % 4 == 0)
33:                     count = 29;
34:                 else
35:                     count = 28;
36:                 if ((year % 100 == 0) & (year % 400 != 0))
37:                     count = 28;
38:             }
39:             return count;
40:         }
41:     }
}
```

该程序使用命令行参数来指定要检查的月份和年份。第一个参数是月份, 它必须是 1~12 的整数, 第二个参数是年份, 它应该是一个 4 位数。

编译该程序后, 请输入如下命令行, 以查看 2008 年 4 月有多少天:

```
java DayCounter 4 2008
```

输出应该是:

```
4/2008 has 30 days.
```

如果运行该程序时, 没有提供任何参数, 将使用默认值 2008 年 1 月, 输出如下所示:

```
1/2008 has 31 days.
```

应用程序 DayCounter 使用 switch 语句来确定天数。这条语句位于程序清单 4.2 的 countDays() 方法中 (第 13~40 行)。

方法 `countDays()` 有两个 `int` 参数: `month` 和 `year`。天数将被存储在变量 `count` 中, 它的初始值为 -1, 然后被替代为正确的计数。

从第 15 行开始的 `switch` 语句使用 `month` 作为条件值。

对于其中的 11 个月份来说, 天数都很容易计算: 1 月、3 月、5 月、7 月、8 月、10 月和 12 月都有 31 天, 而 4 月、6 月、9 月和 11 月都有 30 天。

这 11 个月的计数是在程序清单 4.2 的第 16~39 行中完成的。正如您期望的, 月份从 1 (1 月) 到 12 (12 月) 进行编号。当 `case` 语句与 `month` 的值相同时, 其后所有的语句都被执行, 直到遇到 `break` 语句或到达 `switch` 语句的末尾。

2 月的情况稍微有点复杂, 这是在程序的第 31~37 行处理的。闰年的 2 月有 29 天, 而其他年份的 2 月只有 28 天。闰年必须满足如下条件之一:

- 该年份能被 4 整除, 但不能被 100 整除。
- 该年份可被 400 整除。

正如第 2 章介绍的, 求余运算符 `%` 返回除法运算的余数。几条 `if-else` 语句使用这一条件来判断 2 月份有多少天, 结果取决于年份。

在第 32~35 行中的 `if-else` 语句中, 当年份能够被 4 整除时, 将 `count` 设置为 29, 否则设置为 28。

第 36 和 37 行中的 `if` 语句使用运算符 `&` 来合并两个条件表达式: `year%100!=0` 和 `year & 400!=0`。如果这两个条件都为真, 则 `count` 被设置为 28。

方法 `countDays()` 最后返回 `count` 的值 (第 39 行)。

运行 `DayCounter` 程序时, 第 2~11 行的方法 `main()` 被执行。

在所有的 Java 程序中, 命令行参数被存储在一个 `String` 对象数组中。在 `DayCounter` 中, 这个数组名为 `arguments`。第一个命令行参数被存储在 `argument[0]` 中, 第二个存放在 `argument[1]` 中, 依次向上编号, 直到所有的参数都被存储。如果应用程序运行时没有提供参数, 这个数组也会被创建, 只是不包含任何元素。

第 3 和 4 行创建 `int` 变量 `yearIn` 和 `monthIn`, 用来存储要被检查的年份和月份。它们分别被初始化为 2001 和 2 (表示 2001 年 2 月)。

第 5 行的 `if` 语句使用 `arguments.length` 来确保 `arguments` 数组至少有一个元素。如果有, 则执行第 6 行。

第 6 行调用 `parseInt()`, 并将 `argument[0]` 传递给它。这是类 `Integer` 的一个类方法, 它将 `String` 对象作为参数, 如果该字符串是一个有效的整数, 则将它作为 `int` 形返回。返回的值被存储在 `monthIn` 中。第 7 行执行类似的操作——用 `argument[1]` 作为参数调用 `parseInt()`, 以便设置 `yearIn` 的值。

第 9~11 行显示程序的输出。输出时, 调用了方法 `countDays()`, 并将 `monthIn` 和 `yearIn` 作为参数传递给它, 然后将该方法的返回值显示出来。

#### 注意

至此, 您可能想知道如何从用户那里获取输入, 而不是使用命令行参数来获取。然而, 没有用来读取输入的 `System.out.println()` 方法, 在不使用图形用户界面的情况下, 要读取输入, 必须更深入地了解 Java 中的输入和输出类。这一主题将在第 15 章中介绍。

## 4.5 for 循环

`for` 循环用于重复执行语句, 直到条件得到满足。虽然 `for` 循环通常用于在语句重复次数确定的情况下简化迭代, 但 `for` 循环也可用于几乎任何类型的循环中。

Java 中的 `for` 循环的格式如下:



```
for (initialization; test; increment) {
    statement;
}
```

for 循环的开始包含 3 部分。

- **initialization** 是一个表达式，用来初始化循环的起始状态。如果有循环变量，可在该表达式中进行声明和初始化，例如 `int i = 0`。在 for 循环的这部分中声明的变量是循环中的局部变量；循环执行完毕后，它们将不复存在。可以在这部分初始化多个变量，其中各个表达式之间是用逗号分隔的。语句 `int i = 0, int j = 10` 声明了变量 *i* 和 *j*，它们都是循环中的局部变量。

- **test** 是每次迭代前都将进行的检测，它必须是一个布尔表达式或一个返回布尔值的函数，例如 `i < 10`。如果测试结果为 `true`，则循环继续执行；一旦测试为 `false`，循环将结束。

- **increment** 可以是任何表达式或函数调用。通常，**increment** 用于修改循环变量的值，从而将循环状态逐步引向返回 `false` 的状态，最终结束循环。**increment** 在每次迭代后进行。与 **initialization** 类似，**increment** 也可以包含多个表达式，各个表达式之间用逗号分隔。

for 循环的 **statement** 部分是每次迭代执行的语句。与 if 语句一样，这也可以是单条语句或块语句；前面的例子中使用了一个块，因为这种情况更常见。下面的 for 循环将一个 String 数组的所有元素都设置为 Mr.：

```
String[] salutation = new String[10];
int i; // the loop index variable

for (i = 0; i < salutation.length; i++)
    salutation[i] = "Mr.";
```

其中，变量 *i* 用作循环计数——它计算循环执行的次数。每次循环执行之前，都将该计数值与数组 `salutation` 中的元素数目 `salutation.length` 进行比较。当循环计数值等于或大于 `salutation.length` 时，循环结束。

for 语句的最后一部分是 `i++`。这使得每次循环后，循环计数值都加 1。如果没有这条语句，循环将不会结束。

循环内的语句将数组 `salutation` 的元素设置为 “Mr.”。循环计数用于判断对哪个元素进行修改。

for 循环的任何部分都可以是一条空语句，即不带任何表达式和语句的分号，这样，这部分将被忽略。注意，在 for 循环中使用空语句后，必须在程序的其他地方初始化或递增循环变量（循环计数）。

如果所有的工作都在循环的第 1 行完成了，则 for 循环体也可以是一条空语句。例如，下面的 for 循环找出大于 4000 的第一个质数（它调用了名为 `notPrime()` 的方法，该方法返回一个布尔值，指出 *i* 是不是质数）。

```
for (i = 4001; notPrime(i); i += 2)
    ;
```

对于 for 循环，一个常见的错误是，for 语句以分号结尾：

```
for (i = 0; i < 10; i++);
    x = x * i; // this line is not inside the loop!
```

在这个例子中，第一个分号将结束循环，因此 `x = x * i` 不是循环的一部分。`x = x * i` 只被执行一次，因为它位于 for 循环之外。注意不要在 Java 程序中犯这种错误。

接下来需要做的是使用 for 循环重新编写应用程序 `HalfDollar`，以删除冗余的代码。

最初的代码很长，重复的内容很多，处理的是包含 3 个元素的数组。程序清单 4.3 是更简短、更灵活版本（但输出相同）。

#### 程序清单 4.3 完整的 `HalfLooper.java` 源代码

```
1: class HalfLooper {
2:     public static void main(String[] arguments) {
3:         int[] denver = { 2500000, 2900000, 3500000 };
4:         int[] philadelphia = { 2500000, 2900000, 3800000 };
5:         int[] total = new int[denver.length];
```



```

6:         int sum = 0;
7:
8:         for (int i = 0; i < denver.length; i++) {
9:             total[i] = denver[i] + philadelphia[i];
10:            System.out.format((i + 2003) + " production: %,d%n",
11:                               total[i]);
12:            sum += total[i];
13:        }
14:
15:        System.out.format("Average production: %,d%n",
16:                           (sum / denver.length));
17:    }
18: }

```

该程序的输出如下:

```

2003 production: 5,000,000
2004 production: 5,800,000
2005 production: 7,300,000
Average production: 6,033,333

```

这里使用了一个 for 循环,而不是分别对 3 个数组进行处理。在程序清单 4.3 中,第 8~13 行的循环执行如下操作。

- 第 8 行: 建立循环,并将名为 *i* 的 int 变量作为循环计数。每次循环后,该计数都将加 1,当 *i* 等于或大于数组 *denver* 的元素数目 *denver.length* 时,循环结束。
- 第 9~11 行: 设置循环计数所指定的 *total* 元素的值,然后显示它以及指明年份的文本。
- 第 12 行: 将 *total* 元素的值加到变量 *sum* 中,后者将用于计算年平均值。

使用更通用的循环遍历数组时,程序将能够处理不同长度的数组,给数组元素赋值并显示这些值。

#### 注意

Java 还有一种用于遍历诸如矢量、链表、集合等数据结构中所有元素的 for 循环,这将在第 8 章介绍这些数据结构时一并介绍。

## 4.6 while 和 do 循环

循环类型还有 while 和 do 循环。与 for 循环一样,while 和 do 循环也让一段 Java 代码重复执行,直到指定的条件满足为止。使用 for、while 还是 do 循环,很大程度上说只是一种编程风格而已。

### 4.6.1 while 循环

while 循环用于重复执行一条语句,直到特定条件不为 true。下面是一个例子:

```

while (i < 13) {
    x = x * i++; // the body of the loop
}

```

与关键字 while 一起的是一个布尔表达式,这里为 *i*<13。如果表达式返回 true,while 循环将执行循环体,然后再次对条件进行检测。这一过程将不断重复下去,直到条件为 false。虽然上面的循环使用花括号来构成一个块语句,但它们并非必需的,因为该循环体只有一条语句: *x* = *x* \* *i*++。然而,使用花括号也不会带来任何问题,如果以后需要在循环体内添加其他语句,花括号将是必不可少的。

程序清单 4.4 中的应用程序 ArrayCopier 使用 while 循环将一个 int 数组 (*array1*) 中的元素复制到一个 float 数组 (*array2*) 中,并将每个元素转换为 float 类型。需要注意的是,如果第一个数组中的某个元素值为 1,循环将立刻结束。

程序清单 4.4 完整的 ArrayCopier.java 源代码

```

1: class ArrayCopier {
2:     public static void main(String[] arguments) {
3:         int[] array1 = { 7, 4, 8, 1, 4, 1, 4 };
4:         float[] array2 = new float[array1.length];
5:
6:         System.out.print("array1: [ ");
7:         for (int i = 0; i < array1.length; i++) {
8:             System.out.print(array1[i] + " ");
9:         }
10:        System.out.println("]");
11:
12:        System.out.print("array2: [ ");
13:        int count = 0;
14:        while ( count < array1.length && array1[count] != 1) {
15:            array2[count] = (float) array1[count];
16:            System.out.print(array2[count++] + " ");
17:        }
18:        System.out.println("]");
19:    }
20: }

```

该程序的输出如下：

```
array1: [ 7 4 8 1 4 1 4 ]
array2: [ 7.0 4.0 8.0 ]
```

main() 方法执行的操作如下。

- 第 3 和 4 行声明数组，array1 是一个 int 数组，被初始化成合适的数字。array2 是 float 数组，其长度与 array1 相同，但没有初始值。
- 第 6~10 行用于输出——使用一个 for 循环来遍历 array1，以打印其元素的值。
- 第 13~17 行执行了一些有意思的操作。这些语句在给 array2 赋值（将数字转换为浮点数）的同时，将这些值打印出来。首先声明了一个 count 变量，用于跟踪数组下标。while 循环检测两个条件，它们分别是遍历完 array1 中的所有元素或 array1 元素的值为 1（这正是程序功能描述的一部分）。

可以使用逻辑条件运算符&&来进行检测。使用&&时，仅当两个条件都为 true 时，整个表达式才为 true，如果其中任何一个为 false，则整个表达式为 false，这将结束循环。

该程序的输出表明，array1 中的前 3 个元素都被复制到 array2 中，但第 4 个元素为 1，因此循环结束。如果 array1 没有值为 1 的元素，array2 将包含与 array1 相同的值。如果 while 循环的测试一开始就为 false（例如，如果 array1 的第一个元素为 1），while 循环体将 1 次也不会执行。如果至少需要执行循环 1 次，可以采用下述两种方法之一：

- 在 while 循环外复制该循环体；
- 使用 do 循环（这将在下一节介绍）。

其中，do 循环是更佳解决方案。

## 4.6.2 do...while 循环

do 循环与 while 循环非常类似，主要区别在于检测条件的位置。while 循环在循环执行前检测条件，因此如果首次检测时条件就为 false，则循环体一次也不会被执行。do 循环在检测条件之前，至少执行循环体一次，因此如果首次检测时条件为 false，则循环体已被执行一次了。

这就是向老爸借车和先斩后奏之间的区别。在第一种情况下，如果老爸拒绝，您将借不到车；在第二种情况下，如果老爸拒绝，您已经借过一次了。

下面的例子使用 do 循环不断将一个 long 值加倍，直到它大于 3 万亿：

```
long i = 1;
do {
    i *= 2;
    System.out.print(i + " ");
} while (i < 3000000000000L);
```

在条件 `i < 3000000000000L` 被检测之前，循环体已执行过一次。然后，如果检测结果为 `true`，循环将继续执行；如果为 `false`，循环将结束。请记住，使用 `do` 循环时，循环体至少执行一次。

## 4.7 跳出循环

在所有循环中，当测试条件满足时循环将结束。有时，在循环执行过程中，当发生了某种情况后，需要提早结束循环。在这种情况下，可以使用关键字 `break` 和 `continue`。

前面已经将 `break` 用于 `switch` 中，`break` 结束 `switch` 语句，继续执行程序。用于循环中，关键字 `break` 的功能与此相同：立即结束当前循环。如果在循环中嵌套了循环，将跳到外层循环中，否则执行循环后的语句。

例如，对于前面将 `int` 数组的元素复制到 `float` 数组中，直到数组末尾或数组元素为 1 为止的 `while` 循环，可以在 `while` 循环体内检测后一种情况，并使用 `break` 跳出循环：

```
int count = 0;
while (count < array1.length) {
    if (array1[count] == 1)
        break;
    array2[count] = (float) array2[count++];
}
```

关键字 `continue` 直接进入循环的下次迭代。对于 `do` 和 `while` 循环，这意味着重新回到块语句从头执行；对 `for` 循环，则计算增量表达式，然后执行块语句。当需要在循环内忽略某些特殊的情况时，关键字 `continue` 很有用。对于前面将一个数组复制到另一个数组中的例子，可以测试当前元素是否为 1，在元素为 1 时使用 `continue` 来进入下次迭代，这样结果数组中将不会包含零。注意，由于跳过了第一个数组中值为 1 的元素，因此需要两个跟踪数组的计数器：

```
int count = 0;
int count2 = 0;
while (count++ <= array1.length) {
    if (array1[count] == 1)
        continue;

    array2[count2++] = (float) array1[count];
} >
```

### 4.7.1 标号

`break` 和 `continue` 都有可选的标号，它告诉 Java 从哪里开始继续执行程序。没有标号时，`break` 跳到外层循环或循环后面的语句处。关键字 `continue` 进入下次迭代。使用标号后，`break` 可以跳到循环外的某个位置，`continue` 可以跳到当前循环外的循环中。

要使用标号，请在循环的起始部分前面添加标号和冒号。然后，使用 `break` 或 `continue` 时，在这些关键字后面加上标号的名称，如下所示：

```
out:
for (int i = 0; i < 10; i++) {
    while (x < 50) {
        if (i * x++ > 400)
            break out;
        // inner loop here
    }
    // outer loop here
}
```

在上述代码片段中，标号 `out` 标记的是外层循环。然后，在 `for` 和 `while` 循环中，当特定条件满足

时, `break` 将跳出这两个循环。如果没有标号 `out`, `break` 将跳出内层循环, 并继续执行外层循环。

### 4.7.2 条件运算符

在条件语句中, 除了使用关键字 `if` 和 `else` 外, 还可以使用条件运算符——有时也叫做三元运算符 (ternary operator), 因为它有 3 个操作数。

条件运算符是一个表达式, 即它将返回一个值——这不同于更通用的 `if`, 后者导致语句或语句块被执行。对于短小简单的条件而言, 条件运算符很有用, 如下所示:

```
test ? trueresult : falseresult;
```

`test` 是一个表达式, 它返回 `true` 或 `false`, 就像 `if` 语句中的条件测试一样。如果 `test` 为 `true`, 则条件运算符返回 `trueresult` 的值; 如果 `test` 为 `false`, 则返回 `falseresult` 的值。例如, 下面的语句检测 `myScore` 和 `yourScore` 的值, 返回较大的一个, 并将它赋给变量 `ourBestScore`:

```
int ourBestScore = myScore > yourScore ? myScore : yourScore;
```

条件运算符的这种用法与下面 `if-else` 语句等价:

```
int ourBestScore;
if (myScore > yourScore)
    ourBestScore = myScore;
else
    ourBestScore = yourScore;
```

条件运算符的优先级很低——通常在所有子表达式计算完毕后才被计算。在优先级上, 唯一比它低的运算符是赋值运算符。为重温运算符优先级, 请参阅第 2 章中的表 2.6。

#### 警告

三元运算的主要好处是, 供经验丰富的程序员创建复杂的表达式。它的功能可以用简单的 `if-else` 语句来实现, 因此在初学这门语言时, 没有必要使用该运算符。这里之所以介绍, 主要原因在于您可能在其他 Java 程序员编写的源代码中遇到它。

## 4.8 总结

了解数组、循环和逻辑后, 便可以让计算机决定是否不断地显示数组的内容。

您学习了如何声明数组变量、将对象赋给它以及访问和修改数组元素。使用 `if` 和 `switch` 条件语句, 可以根据布尔测试来决定执行程序的哪部分。您学习了 `for`、`while` 和 `do` 循环, 它们都能够让程序的一部分不断执行, 直到满足给定的条件为止。

有必要重申的是: 您将在 Java 程序中频繁地使用这些特性。

## 4.9 问与答

问: 我在 `if` 的块语句中声明了一个变量。当 `if` 完成后, 该变量的定义便消失了。它到哪里去了?

答: 从技术上说, 块语句构成了一个新的词法作用域 (lexical scope)。这意味着在块内声明的变量仅在该块内可见和可用。当该块执行完毕后, 其中声明的所有变量都将消失。比较好的办法是在最外层的块中声明所需的变量——通常是块语句的开头。例外情况是非常简单的变量, 例如, 对 `for` 循环的循环计数应在 `for` 循环的第一行进行声明。

问: 为什么不能在 `switch` 语句中对字符串进行检测?

答: 在 Java 中, 字符串是对象, 而 `switch` 只能检测基本数据类型 `byte`、`char`、`short` 和 `int`。要对字

字符串进行比较，必须使用嵌套的 if 语句，它能够进行更通用的表达式测试，包括字符串比较。

## 4.10 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 4.10.1 问题

1. 哪种循环在条件表达式被计算之前至少执行循环体语句一次？
  - a. do-while;
  - b. for;
  - c. while。
2. 哪个运算符返回除法运算的余数？
  - a. /;
  - b. %;
  - c. ?。
3. 数组的哪个实例变量可用来确定数组的长度？
  - a. size;
  - b. length;
  - c. MAX\_VALUE。

#### 答案

1. a，在 do-while 循环中，while 条件语句位于循环的末尾。即使其初始值为 false，循环体也将执行一次。
2. b，求模运算符 (%)。
3. b。

### 4.10.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容或使用 Java 编译器进行检测。

对于下述代码

```
public class Cases {
    public static void main(String[] arguments) {
        float x = 9;
        float y = 5;
        int z = (int)(x / y);
        switch (z) {
            case 1:
                x = x + 2;
            case 2:
                x = x + 3;
            default:
                x = x + 1;
        }
        System.out.println("Value of x: " + x);
    }
}
```

当 x 被显示时，其值为多少？

- a. 9.0;



- b. 11.0,
  - c. 15.0,
  - d. 该程序不能通过编译。
- 答案 c

## 4.11 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

- 使用 DayCounter 程序中的 `countDays()` 方法创建一个应用程序，它显示指定年份中的每一天（从 1 月 1 日到 12 月 31 日）。
- 创建一个类，它将 10 个数字对应的单词（one 到 ten）转换为一个 long 值。使用 switch 语句来进行转换，并通过命令行参数获取要转换的单词。



## 第 5 章

# 创建类和方法

如果读者以前使用过其他编程语言，可能被术语“类”的含义所困扰。它看上去与术语“程序”的含义相同，但您对这两者之间的关系不太确定。

在 Java 中，程序由主（main）类以及用于支持主类的其他类组成。支持类包括您可能需要的 Java 2 类库中的类（如 String、Math 等）。

在本章中，随着创建类和方法（它定义了类或对象的行为），您将明白类的含义。本章的内容如下：

- 类定义的组成部分；
- 创建和使用实例变量；
- 创建和使用方法；
- Java 应用程序中的 main() 方法；
- 创建重载方法，名称相同，但定义和特征标不同；
- 创建构造方法，在创建对象时被调用。

### 5.1 定义类

由于前几章中都创建过类，因此现在您应熟悉有关类定义的基础知识。类是通过关键字 class 和名称来定义的，如下所示：

```
class Ticker {  
    // body of the class  
}
```

默认情况下，类将从 Object 派生而来，后者是 Java 类层次结构中所有类的超类。

关键字 extends 用于指定超类，如下面的 Ticker 的子类所示：

```
class SportsTicker extends Ticker {  
    // body of the class  
}
```

### 5.2 创建实例变量和类变量

每当创建类时，都需要定义新类不同于其超类的行为。

行为是通过指定新类的变量和方法来定义的。Java 中有 3 种常用的变量：类变量、实例变量和局部变量。下一节将介绍方法。

#### 5.2.1 定义实例变量

在第 2 章中介绍了如何定义和初始化局部变量，这种变量位于方法定义中。实例变量的声明和定义与局部变量几乎相同，主要区别在于前者位于类定义中。

在方法定义外声明，且没有被关键字 `static` 修饰的变量是实例变量。根据编程习惯，大多数实例变量是在第一行类定义后声明的，但也可以在末尾定义。

程序清单 5.1 是一个简单的类定义，它定义了类 `VolcanoRobot`，这个类是从超类 `ScienceRobot` 派生而来的。

程序清单 5.1 完整的 `VolcanoRobot.java` 源代码

```
1: class VolcanoRobot extends ScienceRobot {
2:
3:     String status;
4:     int speed;
5:     float temperature;
6:     int power;
7: }
```

这个类定义包含 4 个变量。由于这些变量都不是在方法内定义的，因此它们是实例变量。

- *status*: 一个字符串，指出机器人的当前状态（如 `exploring` 或 `returning home`）。
- *speed*: 一个整数，指出机器人的当前移动速率。
- *temperature*: 一个浮点数，指出机器人所处环境的当前温度。
- *power*: 一个整数，指出机器人当前的蓄电池电力。

### 5.2.2 类变量

正如前面的章节介绍的，类变量适用于整个类，而不是被存储在类的各个对象中。

类变量适合用于在同一种类的不同对象之间共享信息或记录类级信息。

在类定义中，使用关键字 `static` 来声明类变量，如下例所示：

```
static int sum;
static final int maxObjects = 10;
```

## 5.3 创建方法

正如第 3 章中介绍的，方法定义了对对象的行为，即在对象被创建时发生的事情以及对象在其生存期内能够执行的各种任务。

本节介绍方法定义和方法的工作原理。下一章将更加详细地介绍一些有关方法的高级内容。

### 5.3.1 定义方法

在 Java 中，方法定义有 4 个基本的部分：

- 方法名；
- 参数列表；
- 方法返回的对象类型或基本数据类型；
- 方法体。

方法定义的前面 2 个部分构成了方法的特征标。

#### 注意

为简化本章的任务，省略了方法定义的两个可选部分：限定符（如 `public` 或 `private`）和关键字 `throws`，后者指出了方法可能引发的异常。有关这些部分的知识，将在第 6 章和第 7 章中介绍。



在其他语言中，方法（可能叫做函数、子程序或过程）的名称足以将其与程序中的其他方法区别开来。

在 Java 中，在同一个类中可以包含多个名称相同，但特征标不同的方法。这叫方法重载，将在下一章中介绍。

下面是一个基本的方法定义：

```
returnType methodName(type1 arg1, type2 arg2, type3 arg3 ...) {  
    // body of the method  
}
```

returnType 是方法的返回值的基本类型或类。它可以是任何一种基本类型、类名或 void（如果方法不返回任何值）。

注意，如果方法返回一个数组对象，则方括号可位于 returnType 或参数列表后面。由于前一种格式更容易阅读，因此本书采用这种格式，如下所示：

```
int[] makeRange(int lower, int upper) {  
    // body of this method  
}
```

方法的参数列表是一组变量声明，它们被放在圆括号内，并用逗号分隔。这些参数是方法体内的局部变量，当方法调用时，将获得它们的值。

方法体内可以包含语句、表达式、对其他对象的方法调用、条件语句、循环等。

除非返回类型被声明为 void，否则方法执行完成后，将返回某种类型的值。必须在方法的出口点使用关键字 return 显式地返回这个值。

程序清单 5.2 是一个类，它定义了一个 makeRange() 方法，该方法接受两个整数——下界和上界——并创建一个数组，该数组中包含上、下界之间的所有整数。

程序清单 5.2 完整的 RangeLister.java 源代码

```
1: class RangeLister {  
2:     int[] makeRange(int lower, int upper) {  
3:         int[] range = new int[(upper-lower) + 1];  
4:  
5:         for (int i = 0; i < range.length; i++) {  
6:             range[i] = lower++;  
7:         }  
8:         return range;  
9:     }  
10:  
11:     public static void main(String[] arguments) {  
12:         int[] range;  
13:         RangeLister lister = new RangeLister();  
14:  
15:         range = lister.makeRange(4, 13);  
16:         System.out.print("The array: [ ");  
17:         for (int i = 0; i < range.length; i++) {  
18:             System.out.print(range[i] + " ");  
19:         }  
20:         System.out.println("]");  
21:     }  
22:  
23: }
```

该程序的输出如下：

The array: [ 4 5 6 7 8 9 10 11 12 13 ]

该类中的 main() 方法通过使用参数 4 和 13 调用方法 makeRange() 来测试它。在第 5~7 行，该方法创建一个空的整型数组，然后使用 4~13 来填充该数组。

### 5.3.2 关键字 this

您可能想在方法体中引用当前对象，即其方法被调用的对象，这样可以使用该对象的实例变量或

将当前对象作为参数传递给其他方法。

在这种情况下，要引用当前对象，可以使用关键字 `this`。

关键字 `this` 指向当前对象，可用于任何可使用对象引用的地方：在句点表示法中，作为方法的参数、作为当前方法的返回值等。下面是一些使用 `this` 的例子：

```
t = this.x;           // the x instance variable for this object

this.resetData(this); // call the resetData method, defined in
                      // this class, and pass it the current
                      // object

return this;          // return the current object
```

在很多情况下，不需要显式地使用关键字 `this`，因为这是默认的。例如，可以通过名称来引用当前类中定义的实例变量和方法调用，因为 `this` 已隐藏在这些引用中。因此，可以将前面的代码重写为如下所示：

```
t = x;                // the x instance variable for this object
resetData(this);      // call the resetData method, defined in this
                      // class
```

#### 注意

引用实例变量时，是否可以省略关键字 `this` 取决于是否在局部作用域中声明了同名的变量。有关这一主题，将在下一节做更详细的介绍。

由于 `this` 是指向当前实例的引用，因此只能在实例方法的定义体内使用它。在类方法（用关键字 `static` 声明的方法）中，不能使用 `this`。

### 5.3.3 变量作用域和方法定义

要使用变量，必须知道其作用域。

作用域是程序的一部分，在其中可以使用某个变量或其他类型的信息。超出作用域后，该变量将不复存在。

在 Java 中声明变量时，便限定了该变量的作用域。例如，局部变量只能在定义它的语句块中使用。实例变量的作用域为整个类，因此可以被类中的任何实例方法使用。

当您引用变量时，Java 从最里面的作用域向外查找其定义。

最里面的作用域可能是一个块，如 `While` 循环的内容。

下一个作用域可能是包含该代码块的方法。

如果在方法中没有找到，将检查类本身。

由于 Java 这种对变量作用域的查找方式，因此您可以在较小的作用域内创建一个变量来隐藏（或取代）该变量的原始值，这可能带来微妙的 bug。

例如，请看下面的 Java 应用程序：

```
class ScopeTest {
    int test = 10;

    void printTest() {
        int test = 20;
        System.out.println("Test: " + test);
    }

    public static void main(String[] arguments) {
        ScopeTest st = new ScopeTest();
        st.printTest();
    }
}
```

在这个类中，有两个名称皆为 `test` 的变量。前者是实例变量，被初始化为 10；后者是局部变量，

值为 20。

方法 `printTest()` 中的局部变量 `test` 隐藏了实例变量 `test`。在方法 `main()` 中调用 `printTest()` 时，它显示 `test` 值为 20，虽然有一个值为 10 的实例变量 `test`。为避免这种问题，可以使用 `this.test` 来引用实例变量，使用 `test` 来引用局部变量，但一种更好的办法是尽量避免相同的变量名和定义。

隐蔽性更强的情况是，在子类中重新定义了超类中已有的变量。这可能导致难以发现的 bug。例如，您可能调用方法来修改一个实例变量的值，但却修改了另一个变量。将对象从一种类转换为另一种类时，可能出现另一种 bug：实例变量的值被神秘地修改，因为从超类而不是子类中获取了那个值。

避免这种情况的最好办法是，了解超类中定义的所有变量，这样可避免重新定义超类中已有的变量。

### 5.3.4 将参数传递给方法

调用接受对象参数的方法时，对象是按引用传递的。您在方法内对该对象所做的任何操作都将影响原来的对象。

请记住，这样的对象包括数组以及数组的对象。将数组传递给方法，并在方法中修改其内容时，将影响原来的数组。另一方面，基本数据类型是按值传递的。

程序清单 5.3 中的 `Passer` 类演示了这一点。

程序清单 5.3 `Passer.java` 的源代码

```
1: class Passer {
2:
3:     void toUpperCase(String[] text) {
4:         for (int i = 0; i < text.length; i++) {
5:             text[i] = text[i].toUpperCase();
6:         }
7:     }
8:
9:     public static void main(String[] arguments) {
10:        Passer passer = new Passer();
11:        passer.toUpperCase(arguments);
12:        for (int i = 0; i < arguments.length; i++) {
13:            System.out.print(arguments[i] + " ");
14:        }
15:        System.out.println();
16:    }
17: }
```

这个应用程序接受一个或更多的命令行参数，并以大写的方式显示它们。下面是运行该程序时的输出示例：

```
java Passer Athos Aramis Porthos
```

```
ATHOS ARAMIS PORTHOS
```

应用程序 `Passer` 使用存储在字符串数组 `arguments` 中的命令行参数。

它创建一个 `Passer` 对象，然后调用方法 `toUpperCase()`，并将数组 `arguments` 作为参数传递给这个方法（第 10~11 行）。

由于传递给方法的是指向该数组对象的引用，因此在第 5 行修改数组元素的值时，将修改原来的元素（而不是它的拷贝）。第 12~14 行通过显示该数组说明了这一点。

### 5.3.5 类方法

类变量和实例变量之间的关系与类方法和实例方法的关系相同。

类方法可被类的任何实例使用，也可被其他类使用。此外，与实例方法不同，调用类方法时，不需要有类的实例。

例如，Java 类库中有一个 `System` 类，它定义了一组可用于显示信息、检索配置信息以及完成其他任务的方法。下面是两个使用其类方法的语句：

```
System.exit(0);

//int now = System.currentTimeMillis();
```

方法 `exit(int)` 关闭应用程序，并指定关闭成功(0)或失败(其他值)的状态码。方法 `currentTimeMillis()` 返回一个 `long` 值，指出从 1970 年 1 月 1 日午夜起，到当前过去的微秒数；当前日期和时间的数字表示。

定义类方法时，需要在方法定义前加上关键字 `static`，就像需要在类变量前加上 `static` 一样。例如，前面使用的类方法 `exit()` 的特征标可能如下：

```
static void exit(int arg1) {
    // body of the method
}
```

Java 支持各种基本类型的包裹类，例如，Java 提供了类 `Integer`、`Float` 和 `Boolean`。使用这些类中定义的方法，可以将对象转换为基本类型或将基本类型转换为对象。

例如，类 `Integer` 中的类方法 `parseInt()` 可以用于字符串。字符串作为参数被传递给该方法，将返回该字符串的 `int` 表示。

下面的语句演示了方法 `parseInt()` 的用法：

```
int count = Integer.parseInt("42");
```

在上面的语句中，`parseInt()` 将 `String` 值“42”作为一个 `int` 值(42)返回，该值被存储在变量 `count` 中。

如果方法名前没有关键字 `static`，则该方法将是实例方法。实例方法只能在对象中运行，而不能在类中运行。在第 1 章中，您创建了一个名为 `checkTemperature()` 的实例方法，用来检测机器人所处环境的温度。

#### 提示

大多数操纵或影响特定对象的方法都应定义为实例方法，那些提供通用功能，但不直接影响特定实例的方法应声明为类方法。

## 5.4 创建 Java 应用程序

了解如何创建类、对象、类变量和实例变量以及类方法和实例方法后，便可以将它们组合起来，构成 Java 程序。

应用程序(application)是能独立运行的 Java 程序。

#### 注意

应用程序不同于小应用程序，后者是作为网页的一部分由支持 Java 的浏览器运行的。

Java 应用程序由一个或多个类构成，根据需要，它可大可小。虽然到现在为止，您所创建的所有 Java 应用程序除了向屏幕或窗口输出一些字符外，几乎没做别的，但您可以创建使用窗口、图形和用户界面元素的 Java 应用程序。

要让 Java 应用程序能够运行，只需一个用作程序入口的类即可。

要成为应用程序的入口类，只需包含 `main()` 方法即可。应用程序运行时，首先执行方法 `main()` 中的代码。

方法 `main()` 的特征标如下：

```
public static void main(String[] arguments) {
    // body of method
}
```

其中各项的含义如下。

- **public** 意味着该方法对其他类和对象也是可用的。方法 `main()` 必须被声明为公有的。第 6 章将更详细地介绍公有和私有方法。

- **static** 意味着方法 `main()` 是一个类方法。
- **void** 意味着方法 `main()` 不返回任何值。
- `main()` 接受一个参数：一个字符串数组。该参数用于存储命令行参数，这将在下一节介绍。`main()` 的方法体包含启动应用程序所需的代码，如初始化变量和创建类实例。

当 Java 执行方法 `main()` 时，`main()` 是一个类方法。程序运行时，并不会自动创建包含 `main()` 的类的实例，如果想将该类作为对象来处理，必须在方法 `main()` 中创建它的一个实例（就像在应用程序 `Passer` 和 `RangeLister` 中那样）。

## 助手类

Java 程序可能只包含一个类（包含方法 `main()` 的类），也可能由好几个类构成（即使是简单的程序，实际上也使用了 Java 2 类库中大量的类）。可以在程序中创建任意数目的类。

### 注意

如果您使用的是 JDK，这些类必须位于环境变量 `CLASSPATH` 列出的文件夹中。

只要是 Java 能够找到的类，程序在运行时就可以使用。然而，只有入口类需要 `main()` 方法。该方法被调用后，接下来将执行程序中使用的各种类和对象中的方法。虽然助手类可以包含 `main()` 方法，但程序运行时，它们将被忽略。

## 5.5 Java 应用程序和命令行参数

由于 Java 应用程序是独立的程序，因此将参数或选项传递给应用程序很有用。在第 4 章中的 `DayCounter` 应用程序中，您就这样做了。

可以使用参数来决定应用程序将如何运行或让通用的应用程序能够操纵不同类型的输入。程序参数有多种用途，如打开调试输入或指定要加载的文件。

### 5.5.1 将参数传递给 Java 应用程序

如何将参数传递给 Java 应用程序随 Java 的运行平台和您使用的虚拟机而异。

使用 JDK 中的 `java` 解释器时，要将参数传递给 Java 程序，应在运行程序时，在命令行中提供参数。例如：

```
java EchoArgs April 450 -10
```

在上面的例子中，将 3 个参数（`April`、`450` 和 `-10`）。传递给了程序请注意参数之间的空格。

对于包含空格的参数，必须用引号将其括起。例如，请看下面的命令行：

```
java EchoArgs Wilhelm Niekro Hough "Tim Wakefield" 49
```

使用引号将 `Tim Wakefield` 括起后，它将被视为一个参数。程序 `EchoArgs` 将收到 5 个参数：`Wilhelm`、`Niekro`、`Hough`、`Tim Wakefield` 和 `49`。使用引号可防止空格被视为参数之间的分隔符；传递给程序并被方法 `main()` 接收后，引号将不被视为参数的一部分。

**警告**

这里的引号不是用来标识字符串。传递给应用程序的每个参数都被存储在一个 String 对象数组中，即使它是一个数字值（如上述示例中的 450、-10 和 49）。

### 5.5.2 在 Java 程序中处理参数

运行应用程序时，如果提供了参数，这些参数将被存储在一个字符串数组中，然后被传递给应用程序的方法 `main()`。再来看一下 `main()` 的特征标：

```
public static void main(String[] arguments) {  
    // body of method  
}
```

其中，`arguments` 是用于存储参数列表的字符串数组，可以按您的喜好给这个数组命名。

在方法 `main()` 中，可以使用这些传递给程序的参数，方法是遍历该参数数组，并采用某种方式来处理它们。例如，程序清单 5.4 中的简单 Java 程序可接受任意数目的参数，并返回这些参数的和与平均值。

程序清单 5.4 完整的 `Averager.java` 源代码

```
1: class Averager {  
2:     public static void main(String[] arguments) {  
3:         int sum = 0;  
4:  
5:         if (arguments.length > 0) {  
6:             for (int i = 0; i < arguments.length; i++) {  
7:                 sum += Integer.parseInt(arguments[i]);  
8:             }  
9:             System.out.println("Sum is: " + sum);  
10:            System.out.println("Average is: " +  
11:                (float)sum / arguments.length);  
12:        }  
13:    }  
14: }
```

第 5 行确保至少给该程序传递了一个参数。这是通过 `length` 来实现的，该实例变量包含数组 `arguments` 中的元素个数。

处理命令行参数时，总是需要执行与此类似的操作；否则，当用户提供的命令行参数少于您所期望的个数时，程序将因 `ArrayIndexOutOfBoundsException` 错误而崩溃。

如果至少传递了一个参数，第 6~8 行的 `for` 循环将遍历数组 `arguments` 中的所有字符串。

由于所有的命令行参数都作为一个 `String` 对象被传递给 Java 应用程序，因此在数学表达式中使用它们之前，必须将它们转换为数值。第 7 行使用了类 `Integer` 的 `parseInt()` 方法，该方法将一个 `String` 对象作为参数，并返回一个 `int` 值。

请在命令行上输入如下命令：

```
java Averager 1 4 13
```

输出如下：

```
Sum is: 18  
Average is: 6.0
```

## 5.6 创建名称相同但参数不同的方法

使用 Java 类库时，常常会遇到有多个同名方法的类。

对于名称相同的方法，是通过下述两个因素进行区分的：

- 参数个数；

- 参数的数据类型或对象。

这两个因素都是方法特征标的一部分。使用多个名称相同但特征标不同的方法被称为重载。

方法重载可以避免使用完全不同的方法来完成几乎相同的任务。重载也使得方法能够根据收到的参数进行不同的操作。

当您调用对象的方法时，Java 将对方法名和参数进行匹配，以确定应执行哪个方法定义。

要创建重载方法，可在同一个类中创建多个不同的方法定义，它们的名称相同，但参数列表不同。不同之处可以是参数数目、参数类型，也可以两者都不同。只要参数列表是独特的，Java 允许对方法进行重载。

#### 警告

在区分重载的方法时，Java 不考虑返回值类型。如果创建两个特征标相同，但返回值不同的方法，类将不能通过编译。此外，方法中每个参数的变量名是无关紧要的，只有参数的数目和类型有关系。

下面的程序创建了一个重载方法。它的开头是一个简单的类定义，定义了一个名为 Box 的类，该类定义了一个矩形，这是通过使用 4 个实例变量 (x1、y1、x2 和 y2) 定义矩形的左上角和右下角坐标来实现的：

```
class Box {
    int x1 = 0;
    int y1 = 0;
    int x2 = 0;
    int y2 = 0;
}
```

当类 Box 的实例被创建时，所有实例变量都被初始化为 0。

实例方法 buildBox() 将这些实例变量设置为相应的值：

```
Box buildBox(int x1, int y1, int x2, int y2) {
    this.x1 = x1;
    this.y1 = y1;
    this.x2 = x2;
    this.y2 = y2;
    return this;
}
```

该方法接受 4 个 int 参数，并返回一个指向 Box 对象的引用。由于参数的名称与实例变量相同，因此该方法中引用实例变量时使用了关键字 this。

该方法可以用来创建矩形，但如果想以另一种方式来定义矩形的大小又该如何办呢？一种方式是使用 Point 对象，而不是坐标，因为 Point 对象包含实例变量 x 和 y。

可以重载 buildBox()：创建该方法的另一个版本，它接受两个 Point 对象作为参数：

```
Box buildBox(Point topLeft, Point bottomRight) {
    x1 = topLeft.x;
    y1 = topLeft.y;
    x2 = bottomRight.x;
    y2 = bottomRight.y;
    return this;
}
```

要让该方法能够运行，必须导入 java.awt.Point 类，这样 Java 编译器才能找到它。

另一种定义矩形的方式是，使用顶角坐标、宽度和高度。

```
Box buildBox(Point topLeft, int w, int h) {
    x1 = topLeft.x;
    y1 = topLeft.y;
    x2 = (x1 + w);
    y2 = (y1 + h);
    return this;
}
```

为完成这个例子，需要创建一个 printBox() 方法和 main() 方法，前者显示矩形的坐标，后者采用各种方式来定义矩形。完整的类定义如程序清单 5.5 所示。



程序清单 5.5 完整的 Box.java 源代码

```
1: import java.awt.Point;
2:
3: class Box {
4:     int x1 = 0;
5:     int y1 = 0;
6:     int x2 = 0;
7:     int y2 = 0;
8:
9:     Box buildBox(int x1, int y1, int x2, int y2) {
10:         this.x1 = x1;
11:         this.y1 = y1;
12:         this.x2 = x2;
13:         this.y2 = y2;
14:         return this;
15:     }
16:
17:     Box buildBox(Point topLeft, Point bottomRight) {
18:         x1 = topLeft.x;
19:         y1 = topLeft.y;
20:         x2 = bottomRight.x;
21:         y2 = bottomRight.y;
22:         return this;
23:     }
24:
25:     Box buildBox(Point topLeft, int w, int h) {
26:         x1 = topLeft.x;
27:         y1 = topLeft.y;
28:         x2 = (x1 + w);
29:         y2 = (y1 + h);
30:         return this;
31:     }
32:
33:     void printBox(){
34:         System.out.print("Box: <" + x1 + ", " + y1);
35:         System.out.println(", " + x2 + ", " + y2 + ">");
36:     }
37:
38:     public static void main(String[] arguments) {
39:         Box rect = new Box();
40:
41:         System.out.println("Calling buildBox with coordinates "
42:             + "(25,25) and (50,50):");
43:         rect.buildBox(25, 25, 50, 50);
44:         rect.printBox();
45:
46:         System.out.println("\nCalling buildBox with points "
47:             + "(10,10) and (20,20):");
48:         rect.buildBox(new Point(10, 10), new Point(20, 20));
49:         rect.printBox();
50:
51:         System.out.println("\nCalling buildBox with 1 point "
52:             + "(10,10), width 50 and height 50:");
53:
54:         rect.buildBox(new Point(10, 10), 50, 50);
55:         rect.printBox();
56:     }
57: }
```

下面是该程序的输出：

```
Calling buildBox with coordinates (25,25) and (50,50):
Box: <25, 25, 50, 50>
```

```
Calling buildBox with points (10,10) and (20,20):
Box: <10, 10, 20, 20>
```

```
Calling buildBox with 1 point (10,10), width 50 and height 50:
Box: <10, 10, 60, 60>
```

可以根据实现类行为的需要，定义任意数目的方法版本。



当有几个方法来完成任务时，在一个方法中调用另一个是一种可以考虑的简化方式。例如，第 17~23 行中的 `buildBox()` 方法可替换为如下更简短的方法：

```
Box buildBox(Point topLeft, Point bottomRight) {  
    return buildBox(topLeft.x, topLeft.y,  
        bottomRight.x, bottomRight.y);  
}
```

该方法中的 `return` 语句调用第 9~15 行中的 `buildBox()` 方法，并将 4 个 `int` 参数传递给它，这样得到的结果相同，但使用的语句更少。

## 5.7 构造方法

在类定义中，也可以定义构造方法，这些方法在对象被创建时自动被调用。

构造方法是在对象被创建（被构造）时调用的方法。

与其他方法不同，构造函数不能被直接调用。使用 `new` 来创建类的新实例时，Java 完成下述三项工作。

- 为对象分配内存。
- 初始化对象的实例变量：赋予初值或设置为默认值（数字为 0、对象为 `null`、布尔值为 `false`、字符值为 `'\0'`）。
- 调用类的构造方法，这可能是几个方法中的一个。

如果类没有定义任何构造方法，则结合使用 `new` 和该类的名称时，仍将创建一个对象。然而，您可能必须设置它的实例变量或调用初始化对象所需的方法。

通过在类中定义构造方法，可以为实例变量设置初值、调用基于这些变量的方法、调用其他对象的方法以及设置对象的初始属性。还可以重载构造方法（就像重载常规方法那样），这样可以根据提供给 `new` 的参数创建出具有特定属性的对象。

### 5.7.1 基本的构造方法

构造方法与常规方法类似，但有 3 个基本区别：

- 名称总是与类相同；
- 没有返回类型；
- 不能使用 `return` 语句来返回一个值。

例如，下面的类使用一个构造方法，根据 `new` 的参数来初始化实例变量：

```
class VolcanoRobot {  
    String status;  
    int speed;  
    int power;  
  
    VolcanoRobot(String in1, int in2, int in3) {  
        status = in1;  
        speed = in2;  
        power = in3;  
    }  
}
```

可以使用下面的语句来创建这个类的对象：

```
VolcanoRobot vic = new VolcanoRobot("exploring", 5, 200);
```

这样，实例变量 `status` 将被设置为 `exploring`，`speed` 被设置为 5，`power` 被设置为 200。

### 5.7.2 调用另一个构造方法

如果某个构造方法的部分行为与已有的构造方法相同，则可以在该构造函数中调用已有的构造函数

数。Java 提供了一种特殊的语法来完成这项工作，可用下面的代码来调用当前类中的构造方法：

```
this(arg1, arg2, arg3);
```

用于调用构造方法时，this 的用法与用于访问当前对象的变量时类似。在上面的语句中，this() 中的参数是用于构造方法的。

例如，来看一个简单的类，它使用圆心坐标 (x, y) 和半径长度定义了一个圆。类 Circle 可能有两个构造函数：一个定义了半径，另一个将半径设置为默认值 1：

```
class Circle {
    int x, y, radius;

    Circle(int xPoint, int yPoint, int radiusLength) {
        this.x = xPoint;
        this.y = yPoint;
        this.radius = radiusLength;
    }

    Circle(int xPoint, int yPoint) {
        this(xPoint, yPoint, 1);
    }
}
```

Circle 的第二个构造函数只接受圆心的 x 和 y 坐标作为参数。由于没有定义半径，因此将使用默认值 1——调用第一个构造函数，并将参数 xPoint、yPoint 和整数字面量 1 作为参数传递给它。

### 5.7.3 重载构造方法

与常规方法一样，构造方法也能接受不同数目和类型的参数。这让您能够使用所需属性来创建对象或让对象使用不同类型的输入计算出对应的属性。

例如，前面的 Box 类中定义的方法 buildBox() 可用作构造方法，因为它们被用来将对象的实例变量初始化为合适的值。因此，您可以创建一个构造函数，而不是最初定义的方法 buildBox()（它接受 4 个表示对角坐标的参数）。

程序清单 5.6 是一个新的类：Box2，其功能与原来的 Box 相同，只是它重载的是构造方法，而不是 buildBox() 方法。

程序清单 5.6 完整的 Box2.java 源代码

```
1: import java.awt.Point;
2:
3: class Box2 {
4:     int x1 = 0;
5:     int y1 = 0;
6:     int x2 = 0;
7:     int y2 = 0;
8:
9:     Box2(int x1, int y1, int x2, int y2) {
10:         this.x1 = x1;
11:         this.y1 = y1;
12:         this.x2 = x2;
13:         this.y2 = y2;
14:     }
15:
16:     Box2(Point topLeft, Point bottomRight) {
17:         x1 = topLeft.x;
18:         y1 = topLeft.y;
19:         x2 = bottomRight.x;
20:         y2 = bottomRight.y;
21:     }
22:
23:     Box2(Point topLeft, int w, int h) {
24:         x1 = topLeft.x;
25:         y1 = topLeft.y;
26:         x2 = (x1 + w);
```

```

27:     y2 = (y1 + h);
28: }
29:
30: void printBox() {
31:     System.out.print("Box: <" + x1 + ", " + y1);
32:     System.out.println(", " + x2 + ", " + y2 + ">");
33: }
34:
35: public static void main(String[] arguments) {
36:     Box2 rect;
37:
38:     System.out.println("Calling Box2 with coordinates "
39:         + "(25,25) and (50,50):");
40:     rect = new Box2(25, 25, 50, 50);
41:     rect.printBox();
42:
43:     System.out.println("\nCalling Box2 with points "
44:         + "(10,10) and (20,20):");
45:     rect = new Box2(new Point(10, 10), new Point(20, 20));
46:     rect.printBox();
47:
48:     System.out.println("\nCalling Box2 with 1 point "
49:         + "(10,10), width 50 and height 50:");
50:     rect = new Box2(new Point(10, 10), 50, 50);
51:     rect.printBox();
52: }
53: }
54: }

```

该应用程序的输出与程序清单 5.5 所示的 Box 应用程序相同。

## 5.8 覆盖方法

当您调用对象的方法时，Java 将在该对象的类中查找方法的定义。如果没有找到，则将方法调用沿类层次向上传递，直到找到方法的定义为止。方法继承让您能够在子类中重复定义和使用方法，而无需复制代码。

然而，有时候，您可能希望对象对方法调用做出响应，但行为不同。在这种情况下，可以覆盖该方法。要覆盖方法，可在子类中定义一个特征标与超类相同的方法。这样，当该方法被调用时，将找到并执行子类的方法，而不是超类的方法。

### 5.8.1 创建覆盖现有方法的方法

要覆盖方法，只需在子类中创建一个特征标（名称、返回值和参数列表）与超类相同的方法即可。由于 Java 将执行它找到的第一个与特征标匹配的方法定义，因此新的特征标隐藏了原来的方法定义。

下面是一个简单的例子，程序清单 5.7 包含两个类——Printer 和 SubPrinter，前者包含一个名为 printMe() 的方法，该方法显示有关这个类的对象的信息，后者是 Printer 的子类，它新增了实例变量 z。

程序清单 5.7 完整的 Printer.java 源代码

```

1: class Printer {
2:     int x = 0;
3:     int y = 1;
4:
5:     void printMe() {
6:         System.out.println("x is " + x + ", y is " + y);
7:         System.out.println("I am an instance of the class " +
8:             this.getClass().getName());
9:     }
10: }
11:

```

```

12: class SubPrinter extends Printer {
13:     int z = 3;
14:
15:     public static void main(String[] arguments) {
16:         SubPrinter obj = new SubPrinter();
17:         obj.printMe();
18:     }
19: }

```

编译上述文件时，将生成两个类文件，而不是预期的一个。因为这个源文件定义了类 `Printer` 和 `SubPrinter`，所以编译器将生成两个类文件。用 Java 解释器运行 `SubPrinter`，将得到如下输出：

```

x is 0, y is 1
I am an instance of the class SubPrinter

```

#### 警告

一定要使用解释器运行 `SubPrinter`，而不是 `Printer`；后者没有 `main()` 方法，因此不能作为应用程序运行。

在 `SubPrinter` 的 `main()` 方法中，创建了一个 `SubPrinter` 对象，并调用了方法 `printMe()`。由于 `SubPrinter` 没有定义该方法，因此 Java 将在 `SubPrinter` 的超类中查找，即首先在 `Printer` 中查找。`Printer` 中有一个 `printMe()` 方法，因此它被执行。不幸的是，这个方法不会显示实例变量 `z`，从上面的输出可以知道这一点。

为解决这种问题，可以在 `SubPrinter` 类中覆盖 `Printer` 中的 `printMe()` 方法，并在其中添加一条显示实例变量 `z` 的语句：

```

void printMe() {
    System.out.println("x is " + x + ", y is " + y +
        ", z is " + z);
    System.out.println("I am an instance of the class " +
        this.getClass().getName());
}

```

### 5.8.2 调用原来的方法

通常，覆盖超类中已经实现的方法的原因有两个：

- 完全替换原来的方法定义；
- 扩展原来的方法，加入其他行为。

覆盖方法并重新定义方法将隐藏原来的方法定义。然而有时候，需要在原来的方法中添加行为，而不是完全替换它，尤其是当原来的方法和覆盖它的方法有部分行为相同时。通过在覆盖的方法中调用原来的方法，只需添加新增的行为。

要调用原来的方法，可以使用关键字 `super`。该关键字将方法调用沿类层次结构向上传递，如下所示：

```

void doMethod(String a, String b) {
    // do stuff here
    super.doMethod(a, b);
    // do more stuff here
}

```

关键字 `super` 类似于关键字 `this`，它表示超类。可以在能够使用 `this` 的任何地方使用它，但 `super` 指的是超类而不是当前对象。

### 5.8.3 覆盖构造函数

从技术上说，构造方法是不能被覆盖的。由于它们的名称总是与当前类相同，构造函数是新创建的，而不是继承而来的。在很多时候，这没问题；当类的构造方法被调用时，所有超类中特征标与此相同的

构造方法也将被调用。因此，类中所有继承而来的部分都将被初始化。

然而，为类定义构造方法时，您可能想修改对象的初始化方式，不但初始化类中新增的变量，而且要修改继承而来的变量的内容。为此，可以显式地调用超类的构造方法，并修改需要修改的变量。

要调用超类中的常规方法，可以使用 `super.methodname(arguments)`。由于构造方法没有可供调用的方法名，因此采用如下格式：

```
super(arg1, arg2, ...);
```

注意，Java 对 `super()` 的用法有特殊的规则：它必须是构造函数定义中的第一条语句。如果构造函数没有显式地调用 `super()`，Java 将自动调用：使用不带参数的 `super()`。

由于对方法 `super()` 的调用必须是第一条语句，因此在覆盖构造函数中，您不能像下面这样做：

```
if (condition == true)
    super(1,2,3); // call one superclass constructor
else
    super(1,2); // call a different constructor
```

与在构造方法中使用 `this()` 类似，`super()` 将调用直接超类的构造方法（这可能会调用其超类的构造函数，依次类推）。注意，要使 `super()` 调用管用，超类中必须有特征标与此相同的构造函数。在您编译源文件时，Java 编译器会检查这一点。

您不必调用超类中特征标与子类的构造函数相同的构造函数，而只需要为需要初始化的值调用构造函数。实际上，可以创建一个这样的类，即它的构造函数的特征标与任何超类的构造函数都完全不同。

程序清单 5.8 是一个名为 `NamedPoint` 的类，它是从 `java.awt` 包中的 `Point` 类派生而来的。`Point` 类只有一个构造函数，该构造函数接受  $x$  和  $y$  坐标作为参数，并返回一个 `Point` 对象。`NamedPoint` 新增了一个实例变量（一个表示名称的字符串），并定义了一个用于初始化  $x$ 、 $y$  和名称的构造函数。

程序清单 5.8 `NamedPoint` 类

```
1: import java.awt.Point;
2:
3: class NamedPoint extends Point {
4:     String name;
5:
6:     NamedPoint(int x, int y, String name) {
7:         super(x,y);
8:         this.name = name;
9:     }
10:
11:     public static void main(String[] arguments) {
12:         NamedPoint np = new NamedPoint(5, 5, "SmallPoint");
13:         System.out.println("x is " + np.x);
14:         System.out.println("y is " + np.y);
15:         System.out.println("Name is " + np.name);
16:     }
17: }
```

该程序的输出如下：

```
x is 5
y is 5
Name is SmallPoint
```

`NamedPoint` 的该构造方法调用 `Point` 的构造方法来初始化 `Point` 的实例变量（ $x$  和  $y$ ）。虽然您自己初始化  $x$  和  $y$  也很容易，但您可能不知道 `Point` 在初始化自身时还执行了哪些操作。因此，比较好的办法是将构造方法沿类层次结构向上传递，以确保一切都被正确地设置。

## 5.9 结束方法

结束方法（`finalizer method`）的功能几乎与构造方法相反。构造方法用来初始化对象，而结束方法

在对象被垃圾收集器删除以释放其占用的内存之前被调用。

结束方法是 `finalize()`。Object 类定义了一个默认结束方法，该方法不执行任何操作。要为类创建结束方法，可使用下面的特征标来覆盖方法 `finalize()`：

```
protected void finalize() throws Throwable {
    super.finalize();
}
```

#### 注意

该方法定义中的 `throws Throwable` 部分指出了该方法调用时可能发生的错误。在 Java 中，错误被称为异常（exceptions），这将在第 7 章更详细地介绍。

在 `finalize()` 的方法体中，应包含需要对该对象执行的清理代码。如果需要的话，还可以调用 `super.finalize()`，让超类释放该对象。

可以在任何时候直接调用方法 `finalize()`，它与其他方法几乎相同。然而，调用 `finalize()` 并不会导致对象作为无用单元被收集。只有删除该对象的所有引用后，才会导致它被标记为删除。

优化 Java 类时，一种减少其占用的内存量的方法是，一旦类和实例变量不再需要，便删除指向它们的引用。要删除引用，将其设置为 `null` 即可。

例如，如果类中有一个名为 `mainPoint` 的变量，它指向一个 `NamePoint` 对象，要释放该对象占用的内存，可使用下述语句：

```
mainPoint = null;
```

结束方法最适合用于优化对象的删除，例如，删除其他对象的引用。然而，需要指出的是，在无用单元收集程序何时调用对象的 `finalize()` 方法方面，Java 解释器没有统一的标准，这可能在指向对象的最后一个引用被删除后很长时间才发生。在大多数情况下，根本不需要使用 `finalize()`。

## 5.10 总结

学习完本章后，您应该对 Java 中的类与 Java 程序之间的关系有比较清楚的认识。

Java 程序都有一个主类，它与其他类进行交互。这也许是 Java 与其他语言之间的一个主要区别。在本章中，结合使用了前面介绍的所有有关创建 Java 类的知识。

- 实例变量和类变量，它们用于存储类和对象的属性。
- 实例方法和类方法，它们定义了类的行为。您学习了如何定义方法——包括方法特征标的组成部分，如何从方法中返回值，如何将参数传递给方法以及如何使用关键字 `this` 来引用当前对象。
- Java 应用程序中的 `main()` 方法以及如何从命令行将参数传递给它。
- 重载方法，重用方法名，但提供不同的参数。
- 构造方法，它定义了对象的初始值和其他起始状态。

## 5.11 问与答

问：在我的类中，有一个名为 `origin` 的实例变量；同时在其中的一个方法中，还有一个名为 `origin` 的局部变量。由于变量的作用域的关系，实例变量被局部变量隐藏了。有什么办法来访问实例变量的值吗？

答：要避免这种问题，最简单的办法是避免局部变量的名称与实例变量相同。否则，可以使用 `this.origin` 来引用实例变量，用 `origin` 来引用局部变量。

问：我创建了两种方法，它们的特征标如下：

```
int total(int arg1, int arg2, int arg3) {...}  
float total(int arg1, int arg2, int arg3) {...}
```

当我编译包含这些方法的类时，Java 编译器报错，但它们的特征标并不同，哪里有问题？

答：在 Java 中，仅当参数列表不同时，即参数数目或参数的类型不同时，才是合法的方法重载。返回类型并非方法特征标的一部分，因此重载方法时不考虑它。仅当调用方法时才考虑：如果两个方法的参数列表完全相同，Java 如何知道该调用哪个呢？

问：我编写了接受 4 个参数的程序，但如果提供的参数太少，它将因为运行时错误而崩溃。这是为什么呢？

答：Java 程序必须检查参数数目和类型，Java 不会检查。如果程序需要 4 个参数，则必须检查用户是否提供了 4 个参数，如果没有，就返回一条错误消息。

## 5.12 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 5.12.1 问题

1. 如果局部变量与实例变量同名，如何在局部变量的作用域内引用实例变量？
  - a. 无法引用，应该将其中一个变量重新命名；
  - b. 在实例变量名前使用关键字 `this`；
  - c. 在实例变量名前使用关键字 `super`。
2. 实例变量是在类的什么地方声明的？
  - a. 任何地方；
  - b. 方法外面；
  - c. 类声明之后，第一个方法之前。
3. 如何将包含空格的参数传递给程序？
  - a. 用引号括起；
  - b. 用逗号将参数分开；
  - c. 用句点将参数分开。

答案

1. b，答案 a 是个好办法，变量名冲突可能导致 Java 程序发生难以查找的错误。
2. b，通常，应在类声明之后，方法声明之前声明实例变量，但只要在所有方法之外就可以。
3. a，被传递给程序后，引号将不是参数的一部分。

### 5.12.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器进行测试。

对于下述代码：

```
public class BigValue {  
    float result;  
  
    public BigValue(int a, int b) {
```

```

        result = calculateResult(a, b);
    }

    float calculateResult(int a, int b) {
        return (a * 10) + (b * 2);
    }

    public static void main(String[] arguments) {
        BiggerValue bgr = new BiggerValue(2, 3, 4);
        System.out.println("The result is " + bgr.result);
    }
}

class BiggerValue extends BigValue {
    BiggerValue(int a, int b, int c) {
        super(a, b);
        result = calculateResult(a, b, c);
    }

    // answer goes here
    return (c * 3) * result;
}

```

为使变量 `result` 的结果为 312.0, 应将 `// answer goes here` 替换为什么样的代码?

- `float calculateResult(int c) {`
- `float calculateResult(int a, int b) {`
- `float calculateResult(int a, int b, int c) {`
- `float calculateResult() {`

答案 c

## 5.13 练习

为巩固本章介绍的知识, 请尝试完成下面的练习。

- 修改第 1 章中的 `VolcanoRobot` 程序, 使之包含构造方法。
- 创建一个表示四维点的 `FourDPoint` 类, 这个类是从 `java.awt` 包中的 `Point` 类派生而来的。





## 第 6 章

# 包、接口和其他类特性

类（模板）用于创建对象，后者能够存储数据、完成任务，实现 Java 语言的各种功能。因此，您必须花大量的时间来学习如何使用类。

本章将进一步介绍有关类的知识，深入介绍如何创建、使用和组织它们，并制定规则，规定其他人可以如何使用它们。

这包括以下主题：

- 控制对方法和变量的访问；
- 将类、方法和变量固定下来，使其值或定义不能派生子类或不能被覆盖；
- 创建抽象的类和方法，将通用行为放到超类中；
- 将多个类组合成包；
- 使用接口来连接类层次中的间隙。

### 6.1 限定符

在前面的章节中，您学习了如何定义类、方法和变量。本章中您将学到的编程技术涉及如何组织类的不同决策和思维方式。所有这些技术都有一个共同的地方，那就是它们都使用了 Java 语言中的特殊限定符关键字。

限定符是关键字，将它们加入到定义中可改变含义。

Java 语言有很多限定符，包括：

- 限定符 `public`、`protected` 和 `private` 用于控制对类、方法和变量的访问；
- 限定符 `static` 用于创建类方法和类变量；
- 限定符 `final` 用于固定（finalize）类、方法和变量的实现；
- 限定符 `abstract` 用于创建抽象类和方法；
- 限定符 `synchronized` 和 `volatile` 用于线程。

要使用限定符，可将其对应的关键字放在要限定的类、方法或变量的定义中。限定符位于语句的最前面，如下例所示：

```
public class Calc extends javax.swing.JApplet {
    // ...
}

private boolean offline;

static final double weeks = 9.5;

protected static final int MEANING_OF_LIFE = 42;

public static void main(String[] arguments) {
    // body of method
}
```

在一条语句中使用多个限定符时，它们的顺序无关紧要，只要位于要限定的元素之前即可。不要将方法的返回类型（如 `void`）看作限定符。

限定符是可选的，通过前面的章节中对一些限定符的使用，您应该意识到了这一点。正如您将看到的，使用限定符的原因很多。

## 控制对方法和变量的访问

在程序中，最常用的限定符是控制对方法和变量的访问的限定符：`public`、`private` 和 `protected`。这些限定符决定了类中的哪些变量和方法对其他类是可见的。

通过控制访问，可以控制其他类将如何使用您的类。类中的有些变量和方法只能在该类中使用，应对与该类进行交互的其他类隐藏它们。这被称为封装：对象控制外部世界对它的了解程度以及如何与它进行交互。

封装防止类中的变量被其他类读取或修改。要使用这些变量，只能通过调用该类的方法。

Java 语言提供了 4 种级别的访问控制：公有、私有、受保护和默认（不使用限定符）。

### 1. 默认

声明变量和方法时，可以不使用任何限定符，如下所示：

```
String version = "0.7a";

boolean processOrder() {
    return true;
}
```

声明变量和方法时，如果没有使用任何访问控制限定符，则它们对同一个包中的其他任何类都是可用的。Java 2 类库被组织成包，如 `java.Swing` 和 `java.util`，前者是窗口类，主要用于图形用户界面编程，后者包含大量的实用工具类。

对于声明时没有使用任何限定符的变量，同一个包中的其他任何类都可以读取或修改它；而对于声明时没有使用任何限定符的方法，同一个包中的其他任何类都可以调用它。除此之外，其他任何类都不能以任何方式访问这些元素。

这种访问控制级别并没有过多地对访问进行控制。当您考虑其他类将如何使用您的类时，这种访问控制的用处不大。

#### 注意

前面的讨论提出了一个问题，即到目前为止，您自己的类在什么包中。正如您将在本章后面看到的，可以使用包声明来使您的类成为包的成员。如果您不这样做，这个类将被放到一个专门的包中，其中的所有类都不属于其他任何包。

### 2. 私有

要完全将方法或变量隐藏起来，不被其他任何类使用，可使用 `private` 限定符。这种变量仅在其所在的类中是可见的。

例如，私有实例变量可被其所属类中的方法使用，但不能被其他任何类的对象使用。私有方法也可被其所属类中的其他方法调用，但不能被其他任何方法调用。这种限制也会影响继承：任何私有变量和私有方法都不能被子类继承。

在下面两种情况下，私有方法很有用：

- 其他类没有理由使用该变量时；
- 其他类以不合适的方式修改该变量将带来严重的后果。

例如，来看一个名为 `CouppnMachine` 的 Java 类，它为一个网上购物站点生成折扣。其中一个名为

winRatio 的变量，可以根据购买量控制折扣比例。可以想象得到，这个变量在网站中占有非常重要的地位。如果它可被其他类修改，则 CouponMachine 的行为将发生剧烈的变化。为防止这种情况的发生，可以将变量 winRatio 声明为私有的。

下面的类使用了私有访问控制：

```
class Logger {
    private String format;

    public String getFormat() {
        return this.format;
    }

    public void setFormat(String format) {
        if ( (format.equals("common")) != (format.equals("combined")) ) {
            this.format = format;
        }
    }
}
```

在这个例子中，类 Logger 的变量 format 是私有的，其他类不能直接检索和设置它的值。

要访问这个变量，只能通过公有方法 getFormat() 和 setFormat(String)，其中前者返回 format 的值，后者设置它的值。

方法 setFormat(String) 包含这样的逻辑：只能将变量 format 设置为 common 或 combined。这演示了将公有方法作为访问实例变量的唯一途径的优点：这些方法能够对如何检索和设置变量的值进行控制。

使用 private 限定符是对象封装自身的主要方式。如果不使用 private 来隐藏变量和方法，将无法控制类被使用的方式。如果不对访问进行控制，其他类将可以自由地修改类的变量，并以任何方式调用其方法。

使用 private 限定符的一个优点是：您可以修改类的实现，而不会影响使用它的用户。如果您想出了一种更好的完成某项工作的方式，可以重新编写类，只要确保 public 方法接受的参数和返回的类型不变即可。

### 3. 公有

在有些情况下，您可能希望类中的方法或变量可供任何类使用。例如，java.awt 包中的类 Color 包含一些常用颜色变量，如 black。图形类想使用黑色时，将使用该变量，因此不应控制对 black 的访问。

类变量通常被声明为公有的。例如，Football 类中一组用于记录得分的变量。变量 TOUCHDOWN 可能等于 6，而变量 FIELDGOAL 可能等于 3，等等。如果变量是公有的，则其他类便可以使用它，如下面的语句所示：

```
if (yard < 0) {
    System.out.println("Touchdown!");
    score = score + Football.TOUCHDOWN;
}
```

限定符 public 使得方法或变量可供其他任何类使用。在前面编写的每个应用程序中，都使用了该限定符，如下面的语句所示：

```
public static void main(String[] arguments) {
    // ...
}
```

应用程序的 main() 方法必须是公有的。否则，java 解释器将不能调用它，以运行该应用程序。由于类的继承性，所有的公有方法和变量都将被子类继承。

### 4. 保护

第 3 种访问控制级别是使方法和变量仅供以下类使用：

- 子类；
- 同一个包中的其他类。

为此，可以使用 `protected` 限定符，如下例所示：

```
protected boolean outOfData = true;
```

#### 注意

您可能会问，这两组类之间有何区别。子类难道不是与超类位于同一个包中吗？并非总是这样。例如，`Applet` 是 `java.awt.Panel` 的子类，但它位于 `javax.swing` 包中。保护访问控制与默认访问控制之间的区别在于：被保护的变量可被子类使用，即使子类与超类位于不同的包中。

如果想让子类更容易实现，这种级别的访问控制将很有用。您的类可能使用一个方法或变量来帮助完成其任务。由于子类继承了大多数相同的行为和属性，因此它也可能需要完成同样的任务。保护访问控制让子类能够使用助手方法或变量，同时防止无关类使用它们。

来看一个名为 `AudioPlayer` 的类，它播放数字音频文件。`AudioPlayer` 有一个名为 `openSpeaker()` 的方法，这是一个内部方法，它与硬件进行交互，让扬声器能够播放声音。对 `AudioPlayer` 之外的其他类而言，`OpenSpeaker()` 并不重要，因此首先想到的是将它声明为私有的。`AudioPlayer` 类的部分代码如下：

```
class AudioPlayer {
    private boolean openSpeaker(Speaker sp) {
        // implementation details
    }
}
```

如果 `AudioPlayer` 不被继承，上述代码将不错。但如果您要创建一个名为 `StreamingAudioPlayer` 的类，它是 `AudioPlayer` 的子类，情况又将如何呢？这个类有可能需要访问 `openSpeaker()` 方法，以便能够覆盖它，并支持流式音频设备。您不希望任何对象都能够使用该方法（因此它不应是公有的），但希望子类能够访问它。

### 5. 比较各种访问控制级别

这些不同的访问控制类型之间的差别不太好理解，尤其是 `protected` 方法和变量。表 6.1 对各种访问控制进行了总结，以帮助读者弄清楚从限制最小（公有）到限制最严格（私有）的各种访问控制之间的差别。

表 6.1 访问控制级别之间的差别

可见性	公有	保护	默认	私有
同一个类中	是	是	是	是
同一个包中的任何类中	是	是	是	否
包外的任何类	是	否	否	否
同一个包中的子类	是	是	是	否
包外的子类	是	是	否	否

### 6. 访问控制和继承

涉及方法的访问控制的最后一个问题是继承。当您创建子类并覆盖方法时，必须考虑原来方法的访问控制。

作为通用的规则，覆盖方法时，新方法的访问控制不能比原来的方法更严格，但可以更松。下面是一些用于继承方法的规则：

- 在超类中被声明为公有的方法在子类中必须也是公有的；
  - 在超类中被声明为保护的方法在子类中可以是保护的或公有的，但不能是私有的；
  - 对于没有访问控制的方法（没有使用限定符），在子类中其访问控制可以更严格。
- 被声明为私有的方法根本不能继承，因此上述规则不适用。

## 7. 存取器方法

在很多情况下，对于类中的实例变量能够存储的值可能有严格的限制。例如，zipCode 变量，在美国，邮政编码必须是 5 位数。

为防止外部类错误地设置变量 zipCode，可以将它声明为私有的，如下面的语句所示：

```
private int zipCode;
```

然而，其他类必须能够设置 zipCode 变量时，又该怎么办呢？在这种情况下，可以让其他类通过存取器方法来访问该私有变量。

存取器方法因其对某些内容的存取权限而得名，如果不通过存取器方法，将无法访问这些内容。通过使用方法来提供对私有变量的访问，可以控制该变量将如何被使用。在有关邮政编码的范例中，可以防止其他类将 zipCode 设置为不正确的值。

通常，有分别用于读取和设置变量的存取器方法。读取方法的名称以 get 开头，而设置方法的名称以 set 开头，如 setZipCode(int) 和 getZipCode(int)。

使用方法来访问实例变量是面向对象编程中一种常用的技术。这提高了类的可重用性，因为它可以防止变量被不正确地使用。

### 注意

Java 类库大量地使用了存取器方法，这些方法采用的格式与这里的 getZipCode() 和 setZipCode(int) 相同。JavaBean 也使用了存取器方法，JavaBean 是一种创建 Java 对象的技术，它创建的对象变量可在集成开发环境中操纵。

## 6.2 静态变量和方法

您已经在程序中使用过限定符 static，它是在第 5 章中介绍的。限定符 static 用于创建类方法和类变量，如下例所示：

```
public class Circle {
    public static float PI = 3.14159265F;

    public float area(float r) {
        return PI * r * r;
    }
}
```

要访问类变量和类方法，可以使用类名和变量（方法）名，并用句点将它们连接起来，如 Color.black 和 Circle.pi；也可以使用类的对象名，但对于类变量和类方法而言，使用类名更好，这样变量或方法的类型将一清二楚。对于实例变量和实例方法，不能通过类名来引用。

下面的语句使用了类变量和类方法：

```
float circumference = 2 * Circle.PI * getRadius();
float randomNumber = Math.random();
```

### 提示

基于与实例变量相同的原因，类变量也将因私有而受益，这样只能通过存取器方法来使用它们。

本章将创建的第一个项目是一个名为 InstanceCounter 的类，如程序清单 6.1 所示，它使用类变量和实例变量来记录创建了多少个这种类的实例。

程序清单 6.1 完整的 InstanceCounter.java 源代码

```
1: public class InstanceCounter {
2:     private static int numInstances = 0;
3:
4:     protected static int getCount() {
5:         return numInstances;
```

```

6:    }
7:
8:    private static void addInstance() {
9:        numInstances++;
10:    }
11:
12:    InstanceCounter() {
13:        InstanceCounter.addInstance();
14:    }
15:
16:    public static void main(String[] arguments) {
17:        System.out.println("Starting with " +
18:            InstanceCounter.getCount() + " instances");
19:        for (int i = 0; i < 500; ++i)
20:            new InstanceCounter();
21:        System.out.println("Created " +
22:            InstanceCounter.getCount() + " instances");
23:    }
24: }

```

该程序的输出如下：

```

Started with 0 instances
Created 500 instances

```

这个例子有很多特性。第 2 行声明了一个名为 `numInstances` 的私有类变量，用于存储实例数。它是一个类变量（用 `static` 声明的），因为实例计数与整个类（而不是某个实例）相关。它还是私有的，这样将只能通过存取器方法检索它的值。

注意对 `numInstances` 的初始化。实例变量在实例创建时被初始化，而类变量在类创建时被初始化。初始化类之前，无法对类或其实例做任何操作，因此上述类将按照计划的那样进行运行。

第 4~6 行创建了一个 `get` 方法，用于取得私有类变量的值。这个方法也被声明为类方法，因为它被用于类变量。方法 `getCount()` 被声明为保护的，而不是公有的，因为只有这个类及其子类对这个值感兴趣，因此对其他类隐藏该方法。

注意，没有用于设置这个值的存取器方法，这是因为仅当新的实例被创建时，这个值才加 1，它不应被设置为随机值。因此没有创建存取器方法，而是创建了一个特殊的名为 `addInstance()` 的私有方法（第 8~10 行），它将 `numInstance` 的值加 1。

第 12~14 行创建了类的构造方法。构造函数在创建新对象时被调用，这使得它成为调用 `addInstance()` 以将该变量的值加 1 的最为合理的地方。

`main()` 方法表明，可以将该类作为 Java 应用程序来运行，并测试其他所有的方法。在方法 `main()` 中，创建了 10 个 `InstanceCounter` 类的实例，然后显示类变量 `numInstances` 的值。

## 6.3 final 类、方法和变量

限定符 `final` 用于类、方法和变量，指出它们将不会被修改。对于类、方法和变量，`final` 的含义各不相同，如下所示：

- `final` 类不能被继承；
- `final` 方法不能被子类覆盖；
- `final` 变量的值不能被修改。

### 6.3.1 变量

`final` 变量常被称为常量变量（或常量），因为它们的值不会改变。

对于变量，限定符 `final` 通常与 `static` 一起使用，这样，该常量将是类变量。对于保持不变的值，没有什么理由让每个对象都存储这个值的一个拷贝，而应将其声明为类变量。



下面是声明常量的例子：

```
public static final int TOUCHDOWN = 6;
static final String TITLE = "Captain";
```

### 6.3.2 方法

**final** 方法是不能被子类覆盖的。在类声明中，使用限定符 **final** 来声明它们，如下例所示：

```
public final void getSignature() {
    // body of method
}
```

将方法声明为 **final** 的最常见的原因是，提高类的运行效率。通常，当 **java** 运行环境（如 **java** 解释器）运行方法时，它将首先在当前类中查找该方法，接下来在其超类中查找，并一直沿类层次向上查找，直到找到该方法为止。这提供了灵活性和开发工作的容易程度，但代价是速度更低。

如果方法是 **final** 的，**Java** 编译器可以将该方法可执行字节码直接放到调用它的程序中。毕竟，该方法不会被子类覆盖而发生变化。

首次开发一个类时，没有理由使用 **final**。然而，如果要使这样的类的执行速度更快，可以将一些方法修改为 **final** 的。如果这样做，子类将无法覆盖它们，因此应三思而行。

**Java 2** 类库将很多常用的方法声明为 **final** 的，这样当程序调用它们时，执行速度将更快。

**注意**

私有方法是 **final** 的，而无需显式地声明，因为不可能在子类中覆盖它们。

### 6.3.3 类

您通过在类的声明中使用限定符 **final** 来将类固定，如下例所示：

```
public final class ChatServer {
    // body of method
}
```

**final** 类不能被继承。与 **final** 方法一样，这以降低灵活性为代价，提高了 **Java** 语言的速度。

您可能会问，使用 **final** 类有什么损失？这说明您还没有继承过 **Java** 类库中的类。很多常用的类都是 **final** 的，如 **java.lang.String**、**java.lang.Math** 和 **java.net.URL**。如果您想创建一个类，其行为与字符串相似，但要做些修改，您不能继承 **String**，并只定义不同的行为，而必须从空白开始。

在 **final** 类中，所有方法将是 **final** 的，因此声明它们时，无需使用限定符。

由于能够将其行为和属性遗传给子类的类更有用，因此您应仔细考虑：将类声明为 **final** 时，是否得大于失。

## 6.4 抽象类和方法

在类层次结构中，类的位置越高，其定义的抽象程度越高。位于类层次结构顶部的类只能定义所有类都有的行为和属性。更具体的行为和属性应在层次结构的更低层中定义。

定义类层次结构的过程中，当您确定通用的行为和属性时，有时可能遇到一些永远不会被实例化的类。这样的类用于定义其子类都有的行为和方法。

这些类被称为抽象类，是使用限定符 **abstract** 来声明的。下面是一个这样的例子：

```
public abstract class Palette {
    // ...
}
```

**java.awt.Component** 就是一个抽象类，它是所有图形用户界面组件的超类。由于很多组件都是从此

个类派生而来的，因此它包含了对这些组件都很有用的方法和变量。然而，并没有这样的通用组件，即可以被加入到用户界面中，从而无需在程序中创建 Component 对象。

抽象类可以包含常规类能够包含的任何东西，这包括构造方法，因为子类可能需要继承这种方法。抽象类也可以包含抽象方法，这时只有方法特征标，而没有实现的方法。这些方法将在抽象类的子类中被实现。抽象类是用限定符 abstract 来声明的。不能在非抽象类中声明抽象方法。如果一个抽象类除抽象方法外什么都没有，则使用接口更合适，这将在本章后面介绍。

## 6.5 包

正如前面指出的，包是一种组织类的方式。包中可以包含任意数量的类，这些类的用途相关或有继承关系。

如果程序较小，且使用的类不多，则根本没有必要了解包。但随着您创建的 Java 程序越来越复杂，将使用大量通过继承彼此相关的类，这是您将它们组织成包的好处。

由于以下几个主要的原因，包很有用。

- 包让您能够将类组织成单元。在硬盘中，您通过文件夹或目录来组织文件和应用程序，而包让您能够将类组织成组，这样每个程序只使用所需的类。
- 包减少了名称冲突带来的问题。随着 Java 类数量的增长，类重名的可能性也不断增加。当您在同一个程序中使用多组类时，可能发生名称冲突，从而导致错误。包让您能够引用所需的类，即使这个类与另一个包中的某个类同名。
- 包让您能够大面积地保护类、变量和方法，而不是分别对每个类进行保护。后面将对此做更详细的介绍。
- 包可用于唯一地标识类。

## 6.6 使用包

本书一直在使用包。每当您使用命令 import 时或通过全名来引用类时（如 java.util.StringTokenizer），都使用了包。

要使用包中的类，可采用下述三种机制之一。

- 如果要使用的类位于包 java.lang 中（如，System 或 Date），可以通过类名来引用它。Java.lang 包中的类将自动被导入到所有的程序中。
- 如果要使用的类位于其他包中，可以通过全名——包括包名（如 java.awt.Font）——来引用它。
- 对于频繁使用的类，可以导入单个类或整个包。类或包被导入后，只需通过名称便可引用相应的类。

没有被指定其所属包的类将被放到一个未命名的默认包中。要引用这种类，只需使用其名称即可。

### 6.6.1 完整的包名和类名

要引用其他包中的类，可以使用全名：包名和类名。以下面的方式使用时，无需将类或包导入：

```
java.awt.Font text = new java.awt.Font()
```

对于程序中只使用一两次的类，使用全名更合适。然而，如果需要使用某个类多次或者包名非常长（包含很多子包），则应该将类导入，这样可以减少输入量。

当您创建自己的包时，应将该包中的所有文件放在同一个文件夹中。包名的各部分分别对应于相应的文件夹。



来看一个名为 BookShipper 的类，它位于 org.cadenhead.library 包中。

这个类的源代码中的第一条语句应如下，它声明了类所属的包的名称：

```
package org.cadenhead.library;
```

编译 BookShipper 类后，必须将其存储在与包名对应的文件夹中。JDK 和其他 Java 工具将在下面几个不同的地方查找文件 org.cadenhead.library.BookShipper.class。

- 执行 Java 命令时所在文件夹的 org\cadenhead\library 子文件夹中。例如，如果在文件夹 C:\J21work 下执行命令，则如果文件 BookShipper.class 位于文件夹 C:\J21work\org\cadenhead\library 中，将成功运行该文件。

- Classpath 中指定的文件夹的 org\cadenhead\library 子文件夹中。
- Classpath 中指定的 Java 存档文件 (JAR) 的 org\cadenhead\library 子文件夹中。

为管理您自己的包以及您使用的其他包，办法之一是将您创建的包的根目录（如 C:\javapackages）加入到 Classpath 中。创建对应于包名的子文件夹后，将包的类文件放到正确的子文件夹中。

### 6.6.2 import 声明

为导入包中的类，可使用 import 声明，就像本书的范例那样。可以导入单个类，如：

```
import java.util.Vector;
```

也可以导入包中的所有类，方法是使用星号 (\*) 代替类名，如下所示：

```
import java.awt.*;
```

在 import 语句中，星号只能用来代替类名，而不能用于导入多个名称类似的包。

例如，Java 2 类库中包含 java.util、java.util.jar 和 java.util.prefs 等包，您不能使用下面的语句来导入这 3 个包：

```
import java.util.*;
```

上述语句只能导入包 java.util。要导入这 3 个包，必须使用下述语句：

```
import java.util.*;
import java.util.jar.*;
import java.util.prefs.*;
```

另外，不能指定类名的一部分（例如，使用 L\* 导入所有以 L 打头的类）。使用 import 声明时，要么导入整个包，要么导入其中的一个类。

import 声明位于文件开头，在任何类定义之前（但在包定义之后，您将在下一节看到这一点）。

分别导入各个类还是导入整个包在很大程度上是一种编码风格。导入整个包并不会降低程序的速度，也不会增加其长度，只有代码中实际使用的那些类才会被加载。必须承认，导入整个包的确使阅读代码的人难以知道类来自于何方。

#### 注意

如果您熟悉 C 或 C++，则可能以为 import 声明类似于 #include，这样由于包含其他文件中的源代码，程序将非常大。在 Java 中，情况并非如此，import 只是告诉 Java 编译器到哪里去查找类，而不会增加编译后的类的大小。

import 语句还可用于通过名称引用类中的常量。

通常，在引用常量时，必须在前面加上类名，如 Color.black、Math.PI 和 File.separator。

通过使用 import static 语句，可以通过更简短的方法引用指定类中的常量。关键字 import static 后面跟一个接口（类）的名称和一个星号，例如：

```
import static java.lang.Math.*;
```

上述语句使得只需使用名称便能够引用 Math 类中的常量，如 E 和 PI。下面是一个简单的类的例子，它利用了这种特性：

```
import static java.lang.Math.*;
```

```
public class ShortConstants {
    public static void main(String[] arguments) {
        System.out.println("PI: " + PI);
        System.out.println("** + (PI * 3));
    }
}
```

### 6.6.3 类名冲突

导入类或包后，通常可以通过类名来引用它，而不用提供包名。但在一种情况下，必须更明确地指出：在不同的包中，有多个名称相同的类。

在进行数据库编程（将在第 18 章中介绍）时便可能发生名称冲突。这种编程可能涉及 `java.util` 和 `java.sql` 包，它们都包含一个名为 `Date` 的类。

如果要在读写数据库中数据的类时用这两个包，可以使用下述语句导入它们：

```
import java.sql.*;
import java.util.*;
```

导入这两个包后，如果像您下面那样在引用 `Date` 类时没有指定包名，将发生编译器错误：

```
Date now = new Date();
```

这是因为 Java 编译器无法判断语句中引用的是哪个 `Date` 类，因此必须像下面那样指定包名：

```
java.util.Date = new java.util.Date();
```

### 6.6.4 Classpath 和类的位置

为了让 Java 能够使用类，它必须能够在文件系统中找到这个类。否则，将显示一条错误消息，指出这个类不存在。Java 使用两个元素来查找类：包名和环境变量 `Classpath` 中列出的目录。

首先，包名对应于文件系统的目录名，因此类 `com.naviseek.Mapplet` 位于目录 `navaiseek` 中，而后者位于 `com` 目录中（即 `com/naviseek/Mapplet.class`）。

Java 将在 `Classpath` 变量列出的 JAR 文件和目录中查找目录。安装 JDK 时，您使用 `Classpath` 变量来指出 Java 类库（一个名为 `tools.jar` 的文件）所在的位置。如果没有提供 `Classpath` 变量，JDK 将只在当前文件夹内查找类。

Java 查找程序中引用的类时，将在这些目录中查找包和类名，如果没有找到类文件，将返回一条错误消息。大多数错误消息 `class not found` 都是由于没有配置好 `Classpath` 变量而造成的。

#### 注意

有关如何在 Windows 或 Linux 系统上针对 JDK 正确地设置 `Classpath`，请参阅附录 A。

要在使用 JDK 编译或运行应用程序时指定 `Classpath`，可使用标记 `-classpath`，然后加上空格，再列出要查找的目录，并用分号（对于 Windows 系统）或冒号（对于 Linux 系统）将目录隔开。例如：

```
javac -classpath /java/lib/tools.jar;/dev/java/root;. Editor.java
```

## 6.7 创建自己的包

在 Java 中，创建包来组织类并不比创建类更复杂。

### 6.7.1 选择包名

首先要确定包名。为包选择什么样的名称取决于您将如何使用这些类。您可能以自己的姓名来给

包命名，也可能根据涉及的 Java 系统部分来命名（如 `graphics` 或 `messaging`）。如果包将以开放源代码的方式发布或是商业产品，则应使用一个能够唯一地标识其作者的包名。

Sun Microsystems 推荐的一种包命名规则是使用 Internet 域名。

为得到包名，将域名中的元素颠倒过来，使域名中最后一部分成为第一部分，然后是倒数第二部分。根据这种规则，由于作者的个人域名为 `cadenhead.org`，因此作者创建的所有包都以 `org.cadenhead` 打头，如 `org.cadenhead.rss`。

这种规则确保其他 Java 开发人员不会提供同名的包，如果它们也遵循这种规则（大多数开发人员确实是这么做的）。

另一个约定是，包名不使用大写字母，以便将其与类名区别开来。例如，内置类 `String` 的全名为 `java.lang.String`，从中很容易区分包名和类名。

### 6.7.2 创建文件夹结构

创建包的第二步是在硬盘上创建一个与包名对应的文件夹结构，名称的每个部分对应于一个不同的文件夹。例如对于包 `org.cadenhead.rss`，需要创建一个名为 `org` 的文件夹，然后在该文件夹中创建一个名为 `cadenhead` 的文件夹，再在 `cadenhead` 中创建一个名为 `rss` 的文件夹，最后将这个包中的类存储到 `rss` 文件夹中。

### 6.7.3 将类加入到包中

创建包的最后一步是将类加入到包中，即在类文件中的 `import` 语句之前添加一条语句——`package` 声明和包名，如下所示：

```
package org.cadenhead.rss;
```

`package` 声明必须是源代码文件的第一行，即位于注释或空行之后，但在 `import` 声明之前。

### 6.7.4 包和类访问控制

前面介绍了用于方法和变量的访问控制限定符，也可以控制对类的访问。

如果没有指定限定符，则类的访问控制为默认级别，即可被同一个包中的其他任何类使用，但在包外不可见，也不可用。通过包保护的类被隐藏在其所在的包中，不能通过名称来导入或引用它。

要让类在包外可见并可导入，可在其定义中加入限定符 `public`，使之成为公有的。

```
public class Visible {
    // ...
}
```

被声明为公有的类可被包外的类导入。

注意，在 `import` 语句中使用星号时，导入的只是包中的公有类。私有类仍被隐藏，只能被包中的其他类使用。

为什么要将类隐藏在包中？原因与在类中隐藏变量和方法相同：创建只有您的实现代码可以使用的类和行为，或者限制程序接口，从而最大限度地降低修改带来的影响。设计类时，就应该对整个包进行全盘考虑，确定哪些类应声明为公有的，哪些类应隐藏。

要创建一个优秀的包，必须定义一组小型的、清晰的公有类和方法，供其他类使用，然后使用隐藏的支持类来实现它们。本章后面将介绍私有类的另一种用途。

## 6.8 接口

与抽象类和抽象方法一样，接口也提供了其他类要实现的行为模板。它们还在类和对象设计方面提供了重大优点，对 Java 的单继承面向对象编程方法提供了有益的补充。

### 6.8.1 单继承存在的问题

在经过深入考虑或经历复杂设计后，您可能发现，对类层次的简化是受到限制的，尤其需要在同一棵继承树的不同分支上同时使用某些行为时。

其他 OOP 语言包含多重继承的概念，可以解决这种问题。通过多重继承，类可以继承多个超类，从而获得全部超类的行为和属性。

这种概念加大了学习和使用编程语言的难度。使用多重继承后，方法调用以及如何组织类层次的问题将复杂得多，同时容易让人迷惑，并带来了多义性。

由于 Java 的目标之一是简单，因此 Java 没有多重继承，采用的是更简单的单继承。

Java 接口是一组抽象行为，可以被混合到任何类中，从而给它添加超类不支持的行为。

Java 接口只包含抽象方法定义和常量——既没有实例变量，也没有方法实现。

在 Java 2 类库中，期望很多完全不同的类都实现某种行为时，都将实现和使用接口。在本章后面，您将使用 Java 类层次中的一个接口——`java.lang.Comparable`。

### 6.8.2 接口和类

虽然类和接口的概念不同，但有很多相同地方。与类一样，接口也是在源文件中声明的，并被编译为 `.class` 文件。在大多数情况下，在可以使用类的地方（用作变量的数据类型、作为强制类型转换的结果等），也可以使用接口。

在本书的所有范例中，几乎都可以将类名替换为接口名。Java 程序员说到“类”时，常常指的是“类或接口”。接口补充并扩展了类的功能，它们几乎被同等对待，但接口不能被实例化：`new` 只能创建非抽象类的实例。

### 6.8.3 实现和使用接口

对于接口，您可以进行两项操作：在类中使用它们和定义自己的接口。接下来在类中使用接口。

要使用接口，可以在类定义中包含关键字 `implements`：

```
public class AnimatedSign extends javax.swing.JApplet
    implements Runnable {
    //...
```

在这个例子中，`javax.swing.JApplet` 是超类，但接口 `Runnable` 扩展了它实现的行为。

由于接口只提供了抽象的方法定义，因此您必须使用同样的特征标来实现这些方法。

实现接口时，必须实现该接口中所有的方法，而不能有选择地实现其中的某些。通过实现接口，您在告诉用户，这个类支持整个接口。

类实现接口后，其子类将继承这些新方法（并可以覆盖或重载它们），就像超类定义了这些方法一样。如果您的类是从实现特定接口的超类派生而来的，则不必在类定义中包含关键字 `implements`。

### 6.8.4 实现多个接口

与单继承的类层次不同，您可以在自己的类中包含任意数目的接口，您的类将实现这些接口中的所有行为。要在类中包含多个接口，只需将它们的名称用逗号分开即可：

```
public class AnimatedSign extends javax.swing.JApplet
    implements Runnable, Observable {
    // ...
}
```

注意，实现多个接口可能将问题复杂化。如果两个接口定义了相同的方法，该如何办呢？可以采用下面3种方式来解决这种问题。

- 如果两个方法的特征标相同，可以在类中实现一个方法，其定义能够满足两个接口。
- 如果方法的参数列表不同，则是一种简单的方法重载：实现两种方法特征标，分别满足各自的接口定义。
- 如果方法的参数列表相同，但返回值不同，则无法创建一个能够满足两个接口的方法。别忘了，仅当参数列表（而不是返回类型）不同时，才能进行方法重载。在这种情况下，试图编译实现这两个接口的类将产生编译器错误，这说明接口设计有缺陷，可能需要重新考虑设计方案。

### 6.8.5 接口的其他用途

几乎在任何可以使用类的地方，都可以使用接口来代替。例如，可以将变量的类型声明为接口：

```
Iterator loop = new Iterator()
```

当变量的类型被声明为接口时，只是意味着它指向的对象必须实现该接口。在这个例子中，由于 loop 包含一个 Iterator 对象，因此这个假设是您可以调用该接口的全部3个方法：hasNext()、next() 和 remove()。

这里的要点是，虽然期望 loop 有3个方法，但可以在实现（甚至创建）合格的类之前，就编写这些代码。在传统的面向对象编程中，必须创建一个带“占位”实现（空方法或只打印无用消息的方法）的类，才能取得相同的效果。

也可以将对象强制转换为接口，就像可以将对象强制转换为其他的类那样。

## 6.9 创建和扩展接口

使用接口后，接下来需要定义自己的接口。接口很像类，声明方式几乎与类相同，也可以被组织成层次结构。然而，声明接口时，必须遵循某些规则。

### 6.9.1 新接口

要创建新的接口，可以这样声明它：

```
interface Expandable {
    // ...
}
```

上述声明几乎与类定义相同，只是使用的是关键字 interface，而不是关键字 class。在接口定义内，可以有方法和变量。

接口内的方法定义是公有和抽象的，可以显式地声明这一点，如果您没有包括这些限定符，它们将被自动转换为公有和抽象的。不能在接口内将方法声明为私有或保护的。

例如，下面的 Expandable 接口有两个方法，其中的一个（expand()）被显式地声明为公有和抽象

的，而另一个 (`contract()`) 被隐式地声明为公有和抽象的。

```
public interface Expandable {
    public abstract void expand(); // explicitly public and abstract
    void contract(); // effectively public and abstract
}
```

注意，与类中的抽象方法一样，接口中的方法也没有方法体。接口中只有方法的特征标，不涉及任何实现。

除了方法外，接口还可以包含变量，但这些变量必须声明为公有、静态和 `final` 的（使之成为常量）。与方法一样，可以显式地将变量声明为公有、静态和 `final` 的，或者不使用这些限定符，将其隐式地声明为这样的。下面是 `Expandable` 的定义，但新增了两个变量：

```
public interface Expandable {
    public static final int increment = 10;
    long capacity = 15000; // becomes public static and final

    public abstract void expand(); // explicitly public and abstract
    void contract(); // effectively public and abstract
}
```

接口必须是公有或包保护的，就像类一样。然而，需要注意的是，如果声明接口时没有使用限定符 `public`，则接口不会自动将其方法转换为公有和抽象的，也不会将其常量转换为公有的。非公有接口的方法和常量也是非公有的，这些方法和常量只能被同一个包中的类和其他接口使用。

与类一样，接口也可以属于某个包。接口还可以导入其他包中的接口和类，就像类一样。

### 6.9.2 接口中的方法

关于接口中的方法，需要注意的是：这些方法被认为是抽象的，可用于任何类，但如何为这些方法定义参数呢？您并不知道什么类将使用它们！答案在于这样一个事实，即可以在任何能够使用类名的地方使用接口名。通过将方法参数定义为接口类型，可以创建通用参数，适用于可能使用该接口的任何类。

来看接口 `Trackable`，它定义了方法 `track()` 和 `quitTracking()`（不带任何参数）。可能还有一个 `beginTracking()` 方法，它接受一个参数：`Trackable` 对象本身。参数应为什么类型呢？应该是实现了接口 `Trackable` 的任何对象，而不能是特定类及其子类。解决办法是，在接口中将参数声明为 `Trackable`：

```
public interface Trackable {
    public abstract Trackable beginTracking(Trackable self);
}
```

然后，在类中实现该方法时，接受通用的 `Trackable` 参数，并将它强制转换为相应的对象：

```
public class Monitor implements Trackable {

    public Trackable beginTracking(Trackable self) {
        Monitor mon = (Trackable) self;
        // ...
    }
}
```

### 6.9.3 扩展接口

与类一样，也可以将接口组织成层次结构。当接口继承另一个接口时，“子接口”将获得“超接口”中声明的所有方法定义和常量。要扩展接口，可使用关键字 `extends`，就像扩展类一样：

```
interface PreciselyTrackable extends Trackable {
    // ...
}
```

注意，与类不同的是，接口层次结构中没有像 `Object` 类那样的接口，它并没有根。接口可以独立存在，也可以继承另一个接口。



另外，不同于类层次，接口层次可以多重继承。例如，一个接口可以继承任意数量的接口（在定义的 `extends` 部分中，使用逗号将它们分开），新接口将包含其所有父接口中的方法和常量。

多重继承时，管理方法名冲突的规则与使用多个接口的类相同。多个只有返回值不同的方法将导致编译错误。

### 6.9.4 创建网上商店

为探究本章前面介绍的所有主题，应用程序 `Storefront` 使用了包、访问控制、接口和封装。该应用程序管理网上商店中的商品，处理两项主要的任务：

- 根据库存量计算每种商品的销售价格；
- 按照销售价格对商品排序。

应用程序 `Storefront` 由两个类构成：`Storefront` 和 `Item`。这些类被组织成一个名为 `org.cadenhead.ecommerce` 的包，因此首先需要定义一个目录结构，用于存储这个包中的类。

JDK 和其他 Java 开发工具会在系统 `Classpath` 列出的文件夹中查找包，同时考虑包的名称。要该项目做好准备工作，新建一个文件夹，它将用作创建的所有包的根文件夹。为此，作者在自己的 Windows XP 系统中创建了文件夹 `C:\dev\java`。

必须将该文件夹添加到 `Classpath` 设置中，有关如何完成这项工作，请参阅附录 A。

创建新包时，在包文件夹中创建一个相应的文件夹。就该项目而言，文件夹结构应为 `org\cadenhead\ecommerce`。

作者在自己的计算机中创建了文件夹 `C:\dev\java\org\cadenhead\ecommerce`，用于存储类文件。

创建用于存储包的文件夹后，将其添加到 `Classpath` 中，再根据程序清单 6.2 创建源文件 `Item.java`。

程序清单 6.2 完整的 `Item.java` 源代码

```

1: package org.cadenhead.ecommerce;
2:
3: import java.util.*;
4:
5: public class Item implements Comparable {
6:     private String id;
7:     private String name;
8:     private double retail;
9:     private int quantity;
10:    private double price;
11:
12:    Item(String idIn, String nameIn, String retailIn, String quanIn) {
13:        id = idIn;
14:        name = nameIn;
15:        retail = Double.parseDouble(retailIn);
16:        quantity = Integer.parseInt(quanIn);
17:
18:        if (quantity > 400)
19:            price = retail * .5D;
20:        else if (quantity > 200)
21:            price = retail * .6D;
22:        else
23:            price = retail * .7D;
24:        price = Math.floor( price * 100 + .5 ) / 100;
25:    }
26:
27:    public int compareTo(Object obj) {
28:        Item temp = (Item)obj;
29:        if (this.price < temp.price)
30:            return 1;
31:        else if (this.price > temp.price)
32:            return -1;

```

```
33:         return 0;
34:     }
35:
36:     public String getId() {
37:         return id;
38:     }
39:
40:     public String getName() {
41:         return name;
42:     }
43:
44:     public double getRetail() {
45:         return retail;
46:     }
47:
48:     public int getQuantity() {
49:         return quantity;
50:     }
51:
52:     public double getPrice() {
53:         return price;
54:     }
55: }
```

编译这个类，并将文件 `Item.class` 移到包 `org.cadenhead.ecommerce` 中。

`Item` 是一个支持类，它表示网上商店出售的产品。其中有一些私有实例变量，用于存储产品的 ID、名称、库存（数量）和零售价格。

由于这个类的所有实例变量都是私有的，因此其他类不能设置或获取它们的值。程序清单 6.2 的第 36~54 行创建了存储器方法，供其他程序获取这些值。其中每个方法都以 `get` 打头，然后是大写的变量名称，这是 Java 类库采用的一种标准。例如，`getPrice()` 返回 `price` 的值。这里没有提供用于设置这些实例变量的方法——将在构造方法中进行设置。

第 1 行指出，`Item` 类位于 `org.cadenhead.ecommerce` 包中。

#### 注意

`cadenhead.org` 是本书作者之一的个人域名，因此该工程遵照了 Sun 的包命名规则：以顶级域名（`org`）开始，然后是开发者的域名（`cadenhead`）和描述包用途的名称（`ecommerce`）。

`Item` 实现了 `Comparable` 接口（第 5 行），这使得对类的对象进行排序更容易。这个接口只有一个方法：`compareTo (Object)`，它返回一个整数。

方法 `compareTo()` 比较类的两个对象：当前对象和作为参数传递该方法的对象。该方法的返回值指出了这个类的对象的排列顺序：

- 如果当前对象应排在另一个对象之前，则返回 -1。
- 如果当前对象应排在另一个对象之后，则返回 1。
- 如果两个对象相等，则返回 0。

在 `compareTo()` 方法中，需要决定根据哪个实例变量来对对象进行排序。第 27~34 行覆盖 `compareTo()` 方法，它根据 `price` 变量进行排序。商品将按价格从高到低排列。

为对象实现 `Comparable` 接口后，可以调用两个类方法来对由这种对象组成的数组、链表或其他集合进行排序。创建 `Storefront.class` 后，您将明白这一点。

第 12~25 行的构造函数 `Item()` 接受 4 个 `String` 对象作为参数，并使用它们来设置实例变量 `id`、`name`、`retail` 和 `quantity`。对于最后两个参数，必须分别使用类方法 `Double.parseDouble()` 和 `Integer.parseInt()` 将其从字符串转换为数值。

实例变量 `price` 的值取决于商品的库存量：

- 如果库存超过 400，`price` 被设置为 `retail` 的 50%（第 18 和 19 行）；



- 如果库存在 201 和 400 之间, price 被设置为 retail 的 60% (第 20 和 21 行);
- 对于其他情况, price 被设置为 retail 的 70% (第 22 和 23 行)。

第 24 行对 price 进行四舍五入, 使之只包含两位或更少的小数, 即将诸如 \$6.92999999999999 的价格修改为 \$6.99。方法 `Math.floor()` 将小数舍入为与之最接近且不大于它的整数, 并将结果作为 `double` 值返回。

编译 `Item.class` 后, 便可以创建一个类, 用于表示出售这些商品的网上商店。请按程序清单 6.3 创建 `Storefront.java` 类。

程序清单 6.3 完整的 `Storefront.java` 源代码

```

1: package org.cadenhead.ecommerce;
2:
3: import java.util.*;
4:
5: public class Storefront {
6:     private LinkedList catalog = new LinkedList();
7:
8:     public void addItem(String id, String name, String price,
9:         String quant) {
10:
11:         Item it = new Item(id, name, price, quant);
12:         catalog.add(it);
13:     }
14:
15:     public Item getItem(int i) {
16:         return (Item)catalog.get(i);
17:     }
18:
19:     public int getSize() {
20:         return catalog.size();
21:     }
22:
23:     public void sort() {
24:         Collections.sort(catalog);
25:     }
26: }

```

要编译 `Storefront.java`, 必须将 `Item` 类存储在与包名 `org.cadenhead.ecommerce` 对应的文件夹中。编译 `Storefront` 类后, 将生成的文件移到 `Item` 类所属的文件夹。

`Storefront.class` 用于管理网上商店中的商品。每种商品都是一个 `Item` 对象, 它们被存储在一个名为 `catalog` 的 `LinkedList` 实例变量中 (第 6 行)。

第 8~13 行的 `getItem()` 方法根据传递它的 4 个参数创建一个新的 `Item` 对象: ID、名称、价格和库存量。创建 `Item` 对象后, 调用 `add()` 方法 (将该 `Item` 对象作为参数), 将它加入到链表 `catalog` 中。

方法 `getItem()` 和 `getSize()` 提供了到 `catalog` 变量中存储的私有信息的接口。第 19~21 行的 `getSize()` 方法调用 `catalog.size()` 方法, 后者返回 `catalog` 中的对象数。

由于链表中的对象像数组和其他数据结构那样被编号, 因此可以使用索引号来读取它们。第 15~17 行的 `getItem()` 方法调用 `catalog.get()`, 并将索引号作为参数值, 它返回链表相应位置中的对象。

第 23~25 行的 `sort()` 方法体现了在类 `Item` 中实现 `Comparable` 接口的好处。类方法 `Collections.sort()` 将对链表和其他数据结构中的对象进行排序, 期间将调用对象的 `compareTo()` 方法确定排列顺序。

编译 `Storefront` 后, 便可以开发程序来实际使用包 `org.cadenhead.ecommerce`。请切换到用于存储本书程序的文件夹 (如 `J21work`), 然后按程序清单 6.4 创建源文件 `GiftShop.java`。

#### 警告

请不要将 `GiftShop.java` 保存到 `com.perfect.ecommerce` 包所在的文件夹中, 因为它并不是这个包的一部分 (其中没有语句 `package org.cadenhead.ecommerce`)。Java 编译器将显示一条错误消息, 然后退出, 因为它不会期望在应用程序 `GiftShop` 所在的文件夹中找到 `Storefront.class`。

程序清单 6.4 完整的 GiftShop.java 源代码

```

1: import org.cadenhead.ecommerce.*;
2:
3: public class GiftShop {
4:     public static void main(String[] arguments) {
5:         Storefront store = new Storefront();
6:         store.addItem("C01", "MUG", "9.99", "150");
7:         store.addItem("C02", "LG MUG", "12.99", "82");
8:         store.addItem("C03", "MOUSEPAD", "10.49", "800");
9:         store.addItem("D01", "T SHIRT", "16.99", "90");
10:        store.sort();
11:
12:        for (int i = 0; i < store.getSize(); i++) {
13:            Item show = (Item)store.getItem(i);
14:            System.out.println("\nItem ID: " + show.getId() +
15:                               "\nName: " + show.getName() +
16:                               "\nRetail Price: $" + show.getRetail() +
17:                               "\nPrice: $" + show.getPrice() +
18:                               "\nQuantity: " + show.getQuantity());
19:        }
20:    }
21: }

```

这个应用程序使用了 org.cadenhead.ecommerce，但不属于这个包。

GiftShop 演示了类 Storefront 和 Item 使之可用的公有接口的每一部分。您可以进行如下操作：

- 创建网上商店；
- 添加商品；
- 按销售价格将商品排序；
- 遍历链表，显示每种商品的信息。

#### 警告

如果将文件 Item.class 和 Storefront.class 保存到 Giftshop.java 所在的文件夹中，则可能无法编译该程序，因为 Java 编译器期望在包文件夹中找到这些文件。请将这些文件移到文件夹 org\cadenhead\ecommerce 中，并在另一个文件夹（如 \J21work）中编译 Giftshop.java。

该程序的输出如下：

```

Item ID: D01
Name: T SHIRT
Retail Price: $16.99
Price: $11.89
Quantity: 90

```

```

Item ID: C02
Name: LG MUG
Retail Price: $12.99
Price: $9.09
Quantity: 82

```

```

Item ID: C01
Name: MUG
Retail Price: $9.99
Price: $6.99
Quantity: 150

```

```

Item ID: C03
Name: MOUSEPAD
Retail Price: $10.49
Price: $5.25
Quantity: 800

```

这些类的很多实现细节都对 GiftShop 和其他可能使用这个包的类隐藏了。

例如，开发 GiftShop 的程序员无需知道 Storefront 使用链表来存放商店中所有商品的数据。如果 Storefront 的开发者后来决定使用另一种数据结构，只要方法 getSize() 和 getItem() 返回预期的值，

GiftShop 仍将能够正常工作。

## 6.10 内部类

至此，您接触类都是某个包的成员：因为使用 `package` 声明指定了包名或使用了默认包。属于某个包的类被称为顶级（top-level）类，Java 最初只支持这种类。

在最新的 Java 版本中，可以在类中定义类，就像它是方法或变量一样。

这种类被称为内部（inner）类。程序清单 6.5 中的应用程序 `SquareTool` 使用了一个名为 `Square` 的内部类来计算并存储浮点数的平方。

程序清单 6.5 完整的 `SquareTool.java` 源代码

```

1: public class SquareTool {
2:     public SquareTool(String input) {
3:         try {
4:             float in = Float.parseFloat(input);
5:             Square sq = new Square(in);
6:             float result = sq.value;
7:             System.out.println("The square of " + input + " is " + result);
8:         } catch (NumberFormatException nfe) {
9:             System.out.println(input + " is not a valid number.");
10:        }
11:    }
12:
13:    class Square {
14:        float value;
15:
16:        Square(float x) {
17:            value = x * x;
18:        }
19:    }
20:
21:    public static void main(String[] arguments) {
22:        if (arguments.length < 1) {
23:            System.out.println("Usage: java SquareTool number");
24:        } else {
25:            SquareTool dr = new SquareTool(arguments[0]);
26:        }
27:    }
28: }

```

编译该应用程序后，可以运行它并用一个浮点数作为参数。例如，在 JDK 中，可以在命令行输入下述命令：

```
java SquareTool 13
```

输出将如下：

```
The square of 13 is 169.0
```

如果您运行该程序时没有提供任何参数，程序将显示如下文本，然后退出。

```
Usage: java SquareTool number
```

在这个应用程序中，`Square` 在功能上与包含在程序主类所在的源文件中的助手类并没有什么区别。唯一的区别是，内部类是在类文件中定义的，因此有以下 2 个优点：

- 内部类对其他所有类都是不可见的，这意味着您不必担心它与其他类发生名称冲突；
- 内部类可以访问在顶级类中的变量和方法，但如果作为一个独立的类，将无法访问。

在很多情况下，内部类是一个短小的类文件，旨在实现某种目的。在应用程序 `SquareTool` 中，由于 `Square` 并没有包含很多复杂的行为和属性，因此适合作为内部类来实现。

内部类的名称与包含它的类的名称相关联，当程序被编译时，被自动指定。例如，对于 `Square` 类，Java 编译器将给它指定名称 `SquareTool$Square.class`。

**警告**

使用内部类时，为使程序可用，必须将所有的.class文件都包含进来。每个内部类都有自己的类文件，必须将这些类文件和顶级类包含进来。

内部类虽然看上去只是一种小小的改进，但实际上对 Java 语言做了重大修改。

内部类的作用域规则与变量类似。除非使用全限定名称，否则在内部类的作用域外，它是不可见的，这有助于在包内构建类。在内部类中，可以通过名称引用包含它的类的元素（类变量和成员变量）以及包含它的代码块中的局部变量。

此外，还可以将顶级类用作另一个顶级类的静态成员。与内部类不同，顶级类不能直接使用其他任何类的实例变量。这种嵌套方式让顶级类能够提供一种包式组织方式，即按逻辑将其包含的顶级类分组。

## 6.11 总结

本章介绍了如何将访问控制限定符用于方法和变量，从而封装对象；还介绍了如何在开发 Java 类和类层次时，使用其他限定符，如 `static`、`final` 和 `abstract`。

为进一步体现开发一套类的效果并更好地使用它们，介绍了如何将类组织成包。这些分组将您的程序更好地组织起来，让您可以与很多公开其代码的 Java 程序员共享这些类。

最后，介绍了如何实现接口和内部类，这两种结构对于设计类层次结构很有帮助。

## 6.12 问与答

问：在任何地方使用存取器方法都不会降低 Java 代码的运行速度吗？

答：并不总是这样。随着 Java 编译器的改进，并能够进行更多的最优化，它们将能够自动提高存取器方法的速度。但如果您很关心速度，可以将存取器方法声明为 `final` 的，在大多数情况下，它们在速度上可以与直接存取实例变量媲美。

问：基于所学的知识，私有的抽象方法和 `final` 抽象方法（类）都不符合逻辑，它们合法吗？

答：不，它们将导致编译错误。要有所作用，抽象方法必须被覆盖，而抽象类必须被继承，但如果它们也是私有或 `final` 的，则这两种操作都将是非法的。

问：有人告诉我，应考虑使用 Ant 来管理 Java 包和编译应用程序。请问 Ant 是做什么的？

答：Apache Ant 是一种用于编译和包装 Java 应用程序的开源工具，它是用可扩展的标记语言 (XML) 和 Java 实现的。使用 Ant，可以创建 XML 文件，指出应如何编译、存档和组织您的类。您可以指定多个目标、流程的时限，并轻松地项目的每个开发阶段生成多个构造文件 (build)。

可从网站 <http://www.apache.org> 下载 Ant，它是为 Apache 管理的 Java 开源项目 Jakarta 开发的。Apache 开发了 Struts、Velocity、Tomcat 以及众多很有用的 Java 类库和技术。

Jakarta 项目极其庞大，需要管理数百个 Java 类、JAR 存档文件和其他文件。Ant 在创建 Tomcat Web 服务器方面很有用，本身已成为一个 Apache 开发项目。它已成为 Java 程序员中最流行的构建 (build) 工具。

## 6.13 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 6.13.1 问题

1. 哪些类将自动导入到您的 Java 类中?
  - a. 没有;
  - b. 存储在 Classpath 指定的文件夹中的类;
  - c. java.lang 包中的类。
2. 根据包的命名约定, 包名的第一部分应是什么?
  - a. 您的姓名和一个句点;
  - b. 您的顶级域名和一个句点;
  - c. 文本 java 和一个句点。
3. 如果您创建了一个子类并覆盖一个 public 方法, 则对该方法可以使用哪些限定符?
  - a. 只能是 public;
  - b. public 或 protected;
  - c. public、protected 或默认访问控制。

#### 答案

1. c, 如果要使用简短的类名 (如 LinkedList), 而不是完整的包名和类名 (如 java.util. LinkedList), 必须导入其他所有包。
2. b, 该约定假设所有的 Java 包开发人员都拥有 (或可以使用) 一个 Internet 域名, 以便人们能够从网上下载包。
3. a, 所有公有方法在子类中仍必须是公有的。

### 6.13.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题, 请回答该问题, 而不要查看本章的内容或使用 Java 编译器进行试验。

对于下面的代码:

```
package org.cadenhead.bureau;
```

```
public class Information {  
    public int duration = 12;  
    protected float rate = 3.15F;  
    float average = 0.5F;  
}
```

以及

```
package org.cadenhead.bureau;
```

```
import org.cadenhead.bureau.*;
```

```
public class MoreInformation extends Information {  
    public int quantity = 8;  
}
```

以及

```
package org.cadenhead.bureau.us;
```

```
import org.cadenhead.bureau.*;
```

```
public class EvenMoreInformation extends MoreInformation {  
    public int quantity = 9;
```



```
EvenMoreInformation() {  
    super();  
    int i1 = duration;  
    float i2 = rate;  
    float i3 = average;  
}
```

在 `EvenMoreInformation` 类中，哪些实例变量是可见的：

- a. `quantity`、`duration`、`rate` 和 `average`;
- b. `quantity`、`duration` 和 `rate`;
- c. `quantity`、`duration` 和 `average`;
- d. `quantity`、`rate` 和 `average`。

答案 b

## 6.14 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 对工程 `Storefront` 进行修改，使其中的每种商品都有一个 `noDiscount` 变量。如果该变量为 `true` 时，则以零售价出售商品。
2. 创建一个 `ZipCode` 类，它使用访问控制来确保其实例变量 `zipCode` 总是一个 5 位或 9 位数。



## 第 7 章

# 异常、断言和线程

这是第一周课程的最后一章，将介绍 Java 中 3 个功能最为强大的元素：线程、异常和断言。

线程是实现了 `Runnable` 接口的对象，能与 Java 程序的其他部分一起运行；异常是用于处理 Java 程序运行阶段可能发生的错误的对象；断言是条件语句和布尔值，指出程序运行正常，提供了另一种检测错误的手段。

线程让程序能够更高效地利用其资源，这是通过将程序中计算密集型部分分离开来，避免其降低程序其他部分的速度来实现的；异常和断言让程序能够识别错误并做出响应，如果可能，异常甚至可以帮助程序校正状态。

本章将首先介绍异常，因为开发线程和断言时需要使用异常。

### 7.1 异常

任何语言的程序员都致力于编写没有漏洞的程序，不会崩溃的程序，能够妥善地处理任何情况并能够在异常情况下进行恢复而不给用户带来过多压力的程序。这种愿望很好，但这样的程序并不存在。

在实际的程序中，错误可能产生于程序员没有预料到的问题、没有足够的时间来测试程序或者在程序员控制范围之外的问题——包括（但不限于）用户输入的数据不正确、文件损坏（因此其中的数据不正确）、无法连接网络、硬设备没有响应、太阳黑子、原因不明的故障等。

在 Java 中，这种奇怪的事件可能导致程序出错，这被称为异常。Java 定义了用于处理异常的特性，包括：

- 如何在代码中处理异常，并妥善地从潜在的问题中恢复；
- 如何告诉 Java 和方法的用户，预期有潜在的异常；
- 检测到异常时，如何引发它；
- 如何使用异常来限制代码，使之更健壮。

在大多数编程语言中，要处理错误条件，都必须完成比处理正常运行的程序更多的工作。可能需要使用复杂的语句结构来处理可能发生的错误。

例如，请看下面的语句，它们可能用于加载磁盘中的文件。由于存在很多异常情况（如磁盘错误、文件找不到等），因此加载文件很容易出现问题。如果程序必须使用文件中的数据才能正常运行，则必须处理这些情况。

下面是一种可能的解决方案：

```
int status = loadTextFile();
if (status != 1) {
    // something unusual happened, describe it
    switch (status) {
        case 2:
            System.out.println("File not found");
            break;
```

```

        case 3:
            System.out.println("Disk error");
            break;
        case 4:
            System.out.println("File corrupted");
            break;
        default:
            System.out.println("Error");
    }
} else {
    // file loaded OK, continue with program
}

```

上述代码使用方法 `loadTextfile()` 来加载文件，该方法是在程序的其他地方定义的。该方法返回一个整数，指出文件被正确地加载（`status=1`）还是发生了错误（`status` 等于任何非 1 的值）。

程序使用 `switch` 语句，根据发生的错误对其进行处理。最终的错误处理代码中并没有对最常见的情况（成功加载文件）进行处理。这里只处理了某些错误，如果程序发生其他错误，将需要更多的嵌套 `if...else` 和 `switch...case` 块。

正如读者看到的，创建更大的系统时，错误管理将非常重要。不同的程序员使用不同的特殊值来处理各种错误，而且没有做很好的说明（如果不是根本没有说明的话）。

管理这些错误的代码常常导致程序的初衷模糊不清，使代码难以阅读和维护。

最后，如果您试图以这种方式来处理错误，将没有简单的办法让编译器采用一致的方式对您调用方法时是否使用了正确的参数进行检查。

虽然前面的例子使用的是 Java 语法，但不必在程序中以这样的方式处理错误。Java 提供了一种更佳的处理异常情况的方式：使用异常类。

异常包括可能对程序造成致命打击的错误，还包含其他一些异常情况。通过管理异常，能够管理错误，并妥善地对其进行处理。

结合使用特殊的语言特性、编译阶段的一致性检查和一组可扩展的异常类，管理 Java 程序中的错误和其他异常情况将容易得多。

有了这些特性，您可以将一个全新的要素加入到类、类层次和整个系统的行为和设计中。类和接口的定义描述了程序在理想情况下的行为。通过将异常处理加入到程序设计中，您可以描述程序在不理想的情况下的行为，让使用这些类的人知道，在这些情况下结果将如何。

### 7.1.1 异常类

至此，您很可能至少遇到过一种 Java 异常：您可能错误地输入了方法名或在代码中犯了某个错误，而导致了问题；也许您运行 Java 应用程序时没有提供所需的命令参数，导致出现错误消息 `ArrayIndexOutOfBoundsException`。

发生异常时，应用程序可能退出，并且屏幕上会显示一组难以理解的错误。这些错误就是异常。程序没有成功完成其工作就退出时，说明引发了异常。异常可能由虚拟机和类引发，也可能是程序故意引发的。

这里使用术语“引发”是因为异常还可以被捕获。捕获异常指的是对异常情况进行处理，以避免程序崩溃，后面将对此做更详细的介绍。

Java 异常系统的核心是异常本身。在 Java 中，异常是对象，即它们是从类 `Throwable` 派生而来的类的实例。当异常被引发时，将创建 `Throwable` 类的一个实例。

`Throwable` 有两个子类：`Error` 和 `Exception`。`Error` 实例是 Java 运行环境（虚拟机）内部的错误，这种错误很少见，但通常是致命的，对于它们，您将无能为力（要么捕获它们，要么引发它们）。提供这个类旨在让 Java 在需要时能够使用它们。

`Exception` 类与编程的关系更为紧密。`Exception` 的子类分两组。



- 运行阶段异常 (RuntimeException 类的子类), 如 `ArrayIndexOutOfBoundsException`、`SecurityException` 和 `NullPointerException`。

- 其他异常, 如 `EOFException` 和 `MalformedURLException`。

运行阶段异常通常是由于代码不够健壮造成的。例如, 如果进行了正确的检查, 以确保代码不超越数组边界, 将不会发生 `ArrayIndexOutOfBoundsException` 异常。除非使用一个之前未被声明为用来存储对象的变量, 否则不会发生 `NullPointerException` 异常。

**警告**

如果无论在任何情况下, 程序都将导致运行阶段异常, 则应在处理异常管理之前修复这些问题。

最后一组异常指出发生了奇怪且无法控制的事情。例如, 如果读取文件内容时跨越了文件尾, 将发生 `EOFExceptions` 异常; 而当 URL 格式不正确 (可能是用户输错了) 时, 将发生 `MalformedURLException` 异常。这组异常中包含一些您创建的异常, 用于指出程序中发生了不正常的情况。

与其他类一样, 异常也被组织成层次结构, 其中超类 `Exception` 表示较通用的错误, 而子类表示更具体的错误。当您在代码中处理异常时, 这种组织结构将更为重要。

大多数异常类 (包括 `Throwable`、`Exception` 和 `RuntimeException`) 位于 `java.lang` 包中。其他包定义了其他异常, 这些异常被用于整个类库中。例如, `java.io` 包定义了一个名为 `IOException` 的通用异常类, 它是 `java.io` 包的输入和输出异常类 (`EOFException` 和 `FileNotFoundException`) 和 `java.net` 包中的网络异常类 (如 `MalformedURLException`) 的父类。

## 7.2 管理异常

知道异常是什么后, 如何在代码中处理异常呢? 在很多情况下, 当您调用使用了可能引发异常的方法时, Java 编译器将要求您对异常进行管理: 您必须在代码中对这些异常进行处理, 否则程序将不能通过编译。本节将介绍一致性检测以及如何使用关键词 `try`、`catch` 和 `finally` 来处理可能发生的异常。

### 7.2.1 异常一致性检测

对 Java 类库使用得越多, 遇到类似于下述的编译器错误 (异常!) 的可能性越大:

```
XMLParser.java:32: Exception java.lang.InterruptedException  
must be caught or it must be declared in the throws clause  
of this method.
```

在 Java 中, 方法可以指出它可能引发的错误类型。例如, 读取文件的方法可能引发 `IOException` 错误, 因此声明这些方法时, 需要使用一个特殊的限定符来指出可能的错误。当您在 Java 程序中使用这些方法时, 必须保护代码不受这些异常的影响。这种规则是由编译器执行的, 编译器以相同的方式对代码进行检测, 以确保您调用方法时提供的参数数目和类型都是正确的。

为什么要进行这种检测呢? 它降低了程序由于致命错误而崩溃的可能性, 因为您将预先知道程序使用的方法可能引发什么样的异常。

为确保处理了所有潜在的问题, 您不必再去仔细阅读有关对象的文档或代码——Java 将为您完成这种检查。另一方面, 如果您定义方法时, 指出了它们可能引发的异常, Java 将命令该方法的用户必须对这些错误进行处理。

### 7.2.2 保护代码和捕获异常

假设您编写代码, 并在编译时遇到异常消息。根据该消息, 您必须捕获这种错误或声明方法必须

引发它。

首先捕获可能的异常，这需要完成两项工作：

- 将包含可能引发异常的方法的代码放在 try 块中；
- 在 catch 块中对异常进行处理。

try 和 catch 的意思是“执行该代码块可能引发异常。如果它执行正常，则继续执行程序；否则，捕获异常并对它进行处理。”

您已经见过 try 和 catch。在第 6 章中，使用 String 创建一个整数时使用了如下代码：

```
try {
    float in = Float.parseFloat(input);
} catch (NumberFormatException nfe) {
    System.out.println(input + " is not a valid number.");
}
```

上述语句中发生的情况如下：类方法 Float.parseFloat() 可能引发 NumberFormatException 异常，这表明线程由于某种原因而被中断。

为处理这种异常，将 parseFloat() 放在了 try 块中，并建立了一个与之相关联的 catch 块。该 catch 块将接收 try 中引发的任何 NumberFormatException 对象。

catch 子句的圆括号中的内容类似于方法定义中的参数列表，其中包含将捕获的异常类和变量名。在 catch 块中，可以使用该变量名来引用异常对象。

该对象的常见用途是用来调用其 getMessage() 方法。所有的异常类都包含这种方法，它显示详细的错误消息，描述发生的情况。

另一个很有用的方法是 printStackTrace()，它显示导致异常的方法调用序列。

下面是第 6 章中使用的 try...catch 语句的修订版：

```
try {
    float in = Float.parseFloat(input);
} catch (NumberFormatException nfe) {
    System.out.println("Oops: " + nfe.getMessage());
}
```

至此，介绍的示例都捕获特定类型的异常。由于异常类被组织成层次结构，同时可以在期望超类的地方使用子类，因此可以在一条 catch 语句中捕获一组异常。

例如，编写处理文件输入/输出、Internet 服务器的程序时，需要处理多种类型的 IOException 异常（IO 表示输入/输出）。该异常有两个子类：EOFException 和 FileNotFoundException。通过捕获 IOException，也可以捕获其子类的实例。

要捕获多种没有任何继承关系的异常，可以将多个 catch 块用于同一个 try 块，如下所示：

```
try {
    // code that might generate exceptions
} catch (IOException ioe) {
    System.out.println("Input/output error");
    System.out.println(ioe.getMessage());
} catch (ClassNotFoundException cnfe) {
    System.out.println("Class not found");
    System.out.println(cnfe.getMessage());
} catch (InterruptedException ie) {
    System.out.println("Program interrupted");
    System.out.println(ie.getMessage());
}
```

使用多个 catch 块时，将执行第一个匹配的 catch 块，并忽略其他 catch 块。

#### 警告

在 catch 块中使用 Exception 超类，并在后面的多个 catch 块中使用 Exception 的子类时，可能发生意外的问题。例如，输入/输出异常 IOException 是文件尾异常 EOFException 的超类。如果将 IOException 块放在 EOFException 块前面，该超类将不会捕获任何异常。

### 7.2.3 finally 子句

假设不管发生什么情况，也无论异常是否被引发，代码中的一些操作都必须执行。这通常是释放外部资源，关闭打开的文件等。

虽然可以同时将执行这些操作的代码放在 catch 块中和块外，但这需要将相同的代码放在两个不同的地方，因此编程时应尽可能避免这样做。

相反，应将这种代码放在 try...catch 块中的可选部分 finally 中：

```
try {
    readTextFile();
} catch (IOException ioe) {
    // deal with IO errors
} finally {
    closeTextFile();
}
```

本章的第一个项目演示如何在方法中使用 finally 语句。

程序清单 7.1 中的应用程序 HexReader 读取两位的十六进制数序列，并显示相应的十进制值。要读取的序列有 3 个：

- 000A110D1D260219,
- 78700F1318141E0C,
- 6A197D45B0FFFFFF.

正如第 2 章介绍的，十六进制是一个基数为十六的计数系统，其中一位数为从 0（十进制 0）到 F（十进制 15），两位数为 10（十进制 16）到 FF（十进制 255）。

程序清单 7.1 完整的 HexReader.java 源代码

```
1: class HexReader {
2:     String[] input = { "000A110D1D260219 ",
3:         "78700F1318141E0C ",
4:         "6A197D45B0FFFFFF " };
5:
6:     public static void main(String[] arguments) {
7:         HexReader hex = new HexReader();
8:         for (int i = 0; i < hex.input.length; i++)
9:             hex.readLine(hex.input[i]);
10:    }
11:
12:    void readLine(String code) {
13:        try {
14:            for (int j = 0; j + 1 < code.length(); j += 2) {
15:                String sub = code.substring(j, j+2);
16:                int num = Integer.parseInt(sub, 16);
17:                if (num == 255)
18:                    return;
19:                System.out.print(num + " ");
20:            }
21:        } finally {
22:            System.out.println("***");
23:        }
24:        return;
25:    }
26: }
```

该程序的输出如下：

```
0 10 17 13 29 38 2 25 **
120 112 15 19 24 20 30 12 **
106 25 125 69 176 **
```

第 15 行调用字符串的 substring(int, int)方法来读取 code 中的两个字符，code 是传递给 readLine() 方法的字符串。

**注意**

在 `String` 类的 `substring()` 方法中，选择子串的方式不太直观。第一个参数指出了子串第一个字符的位置，第二个参数指出的并不是最后一个字符的位置，而是最后一个字符的索引加 1。对字符串调用 `substring(2, 5)` 时，将返回第 2 到第 4 个字符。

两个字符的子串包含一个被存储为 `String` 的十六进制数。`Integer` 的类方法 `parseInt` 可用于将该数字转换为一个整数。如果被转换的是十六进制数，则将第二个参数指定为 16；如果是八进制数，则指定为 8。

在应用程序 `HexReader` 中，十六进制数 `FF` 被用来填充序列的末尾，将不显示其对应的十进制值。这是通过使用 `try-finally` 块（第 13~23 行）来实现的。

遇到第 18 行的 `return` 语句时，`try... finally` 块将导致一种不同寻常的情况。您可能认为，`return` 将导致 `readLine()` 方法立即结束。

由于它位于 `try...finally` 块中，因此不管 `try` 块是如何结束的，`finally` 块中的语句都将被执行。因此，最后将显示文本 “\*\*”。

**注意**

除用于处理异常外，`finally` 语句还有其他用途——在循环中，可在语句 `return`、`break` 或 `continue` 后面使用它来执行清理代码，在这种情况下，可使用包含 `finally` 语句但不包含 `catch` 语句的 `try` 语句。

## 7.3 声明可能引发异常的方法

在前面的例子中，介绍了如何处理可能引发异常的方法（通过保护代码并捕获可能发生的异常）。Java 编译器将检查并确保您对方方法的异常进行了处理，但它如何知道需要处理哪些异常呢？

答案是，方法在定义中指出了将可能引发哪些异常。您可以在自己的方法中使用这种机制——实际上，这是一个不错的主意，这样可以告诉其他用户，您的方法可能发生错误。

要指出方法可能引发的异常，可在方法定义中使用特殊子句 `throws`。

### 7.3.1 `throws` 子句

要指出方法中的某些代码可能引发异常，只需在方法的定义后面加上关键词 `throws` 和将引发的异常的名称，例如：

```
public boolean getFormula(int x, int y) throws NumberFormatException {
    // body of method
}
```

如果方法可能引发多种异常，可将它们放在一个 `throws` 子句中，并用逗号分隔：

```
public boolean storeFormula(int x, int y)
    throws NumberFormatException, EOFException {
    // body of method
}
```

注意，与 `catch` 一样，可以使用超类来指出方法可能引发这种异常的所有子类，例如：

```
public void loadFormula() throws IOException {
    // ...
}
```

记住，将关键字 `throws` 加入到方法定义中只是意味着：如果发生错误，该方法可能引发异常，而并不是将发生这种情况。`Throws` 子句只是在方法定义提供了有关潜在异常的信息，并让 Java 能够确保其他人正确地使用该方法。

方法的整体描述是该方法（或类）的设计者和调用者之间的约定（当然，您可能是设计者，也可能是调用者）。

通常，这种描述指出了方法的参数类型、返回类型及其正常情况下的语义。通过使用 `throws`，可以添加一些信息，指出该方法能执行的非常规工作。约定中的这部分内容有助于显式地指出应在程序的哪些地方对异常情况进行处理，从而简化大规模的设计工作。

### 7.3.2 应引发哪些异常

决定指出方法可能引发异常后，必须判断它可能引发哪些异常（并实际引发它们或调用一个将引发它们的方法，下一节将介绍如何引发异常）。

在很多情况下，很容易根据方法的操作判断出来。也许您要创建并引发自己的异常，在这种情况下，您应知道需要引发哪些异常。

您并不需要列出方法可能引发的所有异常；有些异常是由运行环境处理的，这些异常很常见，您不必处理它们。

具体地说，在 `throws` 子句中，无需列出类 `Error`、`RuntimeException`（或其子类）的异常。

这是因为这些异常可能发生在 Java 程序中的任何地方，而且通常这不是由程序员直接造成的。

例如，`OutOfMemoryError` 可能在任何地方、任何时候发生，引起这种异常的原因也很多。这两种异常被称为无需检查的异常。

无需检查的异常是 `RuntimeException` 和 `Error` 的子类，通常是由 Java 运行环境本身引发的。您无需声明您的方法可能引发它们，也无需以其他任何方式对其进行处理。

#### 注意

如果愿意，当然可以在 `throws` 子句中列出这些错误和运行环境异常，但调用这些方法的人可以不处理它们，只有非运行环境异常才必须处理。

所有其他异常都是需要检查的异常，您可以在方法的 `throws` 子句中列出它们。

### 7.3.3 传递异常

有时候，在方法中对某个异常进行处理是不合理的，由调用该方法的方法进行处理更合适。这没有什么问题，将异常传递给调用您的方法的方法进行处理很常见。

例如，来看一个虚构的示例。`WebRetriever` 类使用 Web 地址加载网页，并将其存储在文件中。正如第 17 章中将介绍的，如果不处理 `MalformedURLException`（当 Web 地址的格式不正确时将引发的异常），您将无法使用 Web 地址。

为使用 `WebRetriever`，另一个类调用其构造函数，并将 Web 地址作为参数。如果这个类指定的地址的格式不正确，将引发 `MalformedURLException` 异常。`WebRetriever` 类没有处理这种异常，其定义如下：

```
public WebRetriever() throws MalformedURLException {
    // ...
}
```

这要求使用 `WebRetriever` 的类必须处理 `MalformedURLException` 异常（或使用 `throws` 子句将这些工作推给其他类去完成）。

但有一点是放之四海皆准的：相比于捕获并忽略异常，将异常传递给调用方法更合适。

除了声明引发异常的方法外，还有一种情况，方法定义也可能包含 `throws` 子句：您想使用一个引发异常的方法，但不想捕获或处理该异常。

可以使用 `throws` 子句来声明方法，使之也可能引发合适的异常，而不是在方法体中使用 `try` 和 `catch` 语句。这样处理异常的工作将由调用您的方法的方法去完成。这是另一种告诉 Java 编译器您已经对某个异常进行处理的方式。

使用这种技术，可以创建一个无需使用 try-catch 块便能处理数字格式异常的方法：

```
public void readFloat(String input) throws NumberFormatException {
    float in = Float.parseFloat(input);
}
```

将方法声明为也可能引发异常后，便可以在该方法中使用也可能引发这些异常的其他方法，而无需保护这些代码或捕获异常。

#### 注意

当然，除了将 throws 子句中列出的异常传递出去外，还可以在方法体内使用某种方式来处理其他异常，然后重新引发它，使调用您的方法的方法必须处理它。下一节将介绍如何引发异常。

### 7.3.4 throws 和继承

如果您的方法定义覆盖了超类中包含 throws 子句的方法，则对于覆盖方法如何处理 throws，有一些特殊的规则。新方法的 throws 子句列出的异常并不一定非得与被覆盖的方法相同，这不同于方法特征标的其他部分——必须模仿被覆盖的方法。

由于同只是引发异常相比，新方法对异常进行处理可能更合适，因此新方法可以引发更少的异常，甚至可以不引发任何异常。这意味着下述两个类定义是可行的：

```
public class RadioPlayer {
    public void startPlaying() throws SoundException {
        // body of method
    }
}

public class StereoPlayer extends RadioPlayer {
    public void startPlaying() {
        // body of method
    }
}
```

反过来则不成立：子类方法不能引发其超类方法没有引发的异常。

## 7.4 创建并引发自己的异常

每种异常都有两面：引发和捕获。被捕获前，异常可能被多次传递给多个方法，但最终它将被捕获并得到处理。

异常是由哪个方法引发的呢？异常来自何处？很多异常是由 Java 运行环境或 Java 类中的方法引发的。您可以引发 Java 类库定义的标准异常，也可以创建并引发自己的异常。

### 7.4.1 引发异常

声明您的方法可能引发异常只对该方法的用户和 Java 编译器有用，Java 编译器将确保所有的异常都被处理，但声明本身并不引发异常，您必须在方法中这样做。

要引发异常，必须创建该异常类的实例，然后使用 throw 语句来引发它。

下面的例子使用虚构的 NotInServiceException，它是 Exception 的子类：

```
NotInServiceException nise = new NotInServiceException();
throw nise;
```

只能引发实现了接口 Throwable 的异常。

异常的构造函数可能接受参数，这取决于您使用的异常类。最常见的参数是字符串，它让您能够更详细地描述问题（这对调试很有帮助）。下面是一个例子：

```

NotInServiceException nise = new
    NotInServiceException("Exception: Database Not in Service");
throw nise;

```

异常被引发后，方法将立即结束，而不执行任何其他代码（除 `finally` 中的代码外，如果有这样的块），也不会返回值。如果在调用方法时，没有将该方法调用放在 `try` 块中，并提供相应的 `catch` 块，则程序可能由于引发的异常而退出。

### 7.4.2 创建自己的异常

虽然 Java 类库中有很多异常可供您在自己的方法中使用，但您仍需要创建自己的异常来处理程序中可能出现的各种错误。所幸的是，创建新的异常非常容易。

新异常应继承 Java 层次结构中的某个异常。所有用户创建的异常都应是 `Exception` 而非 `Error` 层次结构的一部分，后者用于表示涉及 Java 虚拟机的错误。找到一个与要创建的异常相近的异常，例如，文件格式不对的异常应是 `IOException` 的子类。如果不能找到与要创建的异常密切相关的异常，可考虑继承 `Exception`，它位于需要检查的异常层次结构的“顶端”（不需要检查的异常是从 `RuntimeException` 派生而来的）。

异常类通常有两个构造函数：第一个接受任何参数，第二个接受一个字符串参数。对于后一种构造函数，应在其中调用 `super()`，以确保该字符串被应用到正确的地方。

除这 3 条规则外，异常类与其他类相同。可以将它们放在独立的源代码文件中，并对其进行编译，就像其他类一样：

```

public class SunSpotException extends Exception {
    public SunSpotException() {}
    public SunSpotException(String msg) {
        super(msg);
    }
}

```

### 7.4.3 组合使用 `throws`、`try` 和 `throw`

如果要组合使用前面介绍的各种方式，该如何办呢？您可能想在方法中对异常进行处理，也可能想将异常传递给调用方法去处理。仅仅使用 `try` 和 `catch` 无法传递异常，而仅仅使用 `throws` 子句则无法处理异常。

要管理异常并把它传递给调用方法，需要使用 3 种机制：`throws` 子句、`try` 语句和 `throw` 语句（显式地引发异常）。

下面是一个使用这种技术的方法：

```

public void readMessage() throws IOException {
    MessageReader mr = new MessageReader();

    try {
        mr.loadHeader();
    } catch (IOException e) {
        // do something to handle the
        // IO exception and then rethrow
        // the exception ...
        throw e;
    }
}

```

由于异常处理程序可以嵌套，所以上述语句管用。应对异常进行处理，但如果觉得它太重要了，则应让调用者中的异常处理程序对其进行处理。

异常将沿着方法调用链向上传递（大多数方法不对其进行处理），最后系统将对未被捕获的异常进行处理：终止程序并打印一条错误消息。

如果能够捕获异常并妥善地处理它，则应该这样做。

## 7.5 何时使用和不使用异常

由于异常的引发、捕获和声明是相关的，且容易混淆，下面就什么时候应该如何做进行小结。

### 7.5.1 什么时候使用异常

如果您的方法调用了另一个可能引发异常的方法，可以采取下述 3 种方式之一：

- 使用 try 和 catch 语句来处理异常；
- 在方法定义添加 throws 子句，将异常沿调用链向上传递；
- 使用 catch 捕获异常，然后使用 throw 重新引发它。

对于引发多种异常的方法，可以分别处理其中的每种异常。例如，您可能捕获其中的一些异常，而让其他异常沿调用链向上传递。

如果您的方法将引发自己的异常，则应使用 throws 子句来声明它。如果您的方法覆盖了引发异常的超类方法，则可以引发同样的异常或该异常的子类，但不能引发其他异常。

最后，如果声明方法时使用了 throws 子句，别忘了在方法体中使用 throws 子句来引发该异常。

### 7.5.2 什么时候不使用异常

在几种情况下，不应使用异常。

首先，对于可能发生但可以通过简单的表达式来避免的异常，则不使用。例如，虽然您可以依赖 `ArrayIndexOutOfBoundsException` 异常来指出超越了数组尾，但可以很容易地使用数组的变量 `length` 来防止这种情况发生。

此外，如果用户输入的数据必须是整数，则与引发异常，并在其他地方进行处理相比，对其进行检测以确保它是整数将好得多。

异常将占用大量的处理时间。简单的测试或一系列的测试比异常处理的速度快得多，可提高程序的效率。仅当您无法控制异常情况时，才应使用异常。

人们常常不由自主地使用异常，并在声明所有的方法时都让它引发所有可能引发的异常。这将使代码更复杂，此外，当其他人使用您的代码时，必须处理方法可能引发的所有异常。

使用异常将导致相关的每个人都需要完成更多的工作。将方法声明为引发少量异常还是大量异常需要折衷考虑，方法引发的异常越多，使用起来就越复杂。应只声明那些发生的可能性较大且对类的整体设计而言比较合理的异常。

### 7.5.3 糟糕的异常使用方式

刚开始使用异常时，您可能想避免编译器错误，这种错误是使用声明了 throws 子句的方法时引起的。虽然在方法中加入空的 catch 子句或 throws 语句是合法的（并且有理由这样做），但不经处理就将异常丢弃将破坏 Java 编译器进行的检查工作。

Java 异常系统被设计成在错误发生后，将告知您这一点。忽略警告并想办法避开将可能使之成为程序中的致命错误，而这种错误原本只需几行代码就可避免。更糟糕的是，在方法中加入 throws 子句来避免异常意味着使用该方法的调用者（即调用链中更上层的对象）必须处理这些异常。这只会使您的方法更难以使用。



这里指出有关异常方面的编译器错误旨在提醒您对这些问题有所考虑。请花些时间来处理可能影响代码的异常。当您在以后的工程和越来越大的程序中重用您的类时，这将带来丰厚的回报。当然，编写 Java 类库时，考虑了这方面的问题，这也是它足够健壮，可用于创建 Java 工程的原因之一。

## 7.6 断言

异常是提高 Java 程序的可靠性的途径之一，断言 (assertion) 是一个表示条件的表达式，程序员认为在程序的特定位置，该条件为真。如果不为真，将导致错误。

关键字 `assert` 后面是一个条件表达式或布尔值，如下所示：

```
assert price > 0;
```

上述 `assert` 语句宣称：变量 `price` 的值大于 0。断言是一种确保程序正确运行的方式，这是通过编写指示正确行为的条件表达式并对其进行测试实现的。

关键字 `assert` 的后面必须包含三项内容之一：结果为 `true` 或 `false` 的表达式、布尔变量或返回布尔值的方法。

如果关键字 `assert` 后面的断言不为真，将引发 `AssertionError` 异常。为使与断言相关联的错误消息更有意义，可在 `assert` 语句中指定一个字符串，如下例所示：

```
assert price > 0 : "Price less than 0.";
```

在这个例子中，如果在该断言被执行时，`price` 小于 0，将引发一个 `AssertionError` 异常，相应的错误消息为 “Price less than 0.”

您可以捕获这种异常，也可以留给 Java 解释器进行处理。下面的例子说明了 `assert` 断言为假时，JDK 的解释器做出的响应：

```
Exception in thread "main" java.lang.AssertionError
    at AssertTest.main(AssertTest.java:14)
```

下面的例子说明了当该 `assert` 语句中包含一条描述性错误消息时的情况：

```
Exception in thread "main" java.lang.AssertionError: Price less than 0.
    at AssertTest.main(AssertTest.java:14)
```

虽然断言是 Java 语言的正式组成部分，但默认情况下，JDK 中的工具并不支持，其他 Java 开发工具也可能不支持。

为使 JDK 支持断言，在运行编译器和解释器时，必须使用命令行参数。

只要使用的是最新的 JDK 版本，包含 `assert` 语句的类便能够被正常编译。

编译器在其生成的类文件中，包含对断言的支持，但必须启用该功能。

启用 JDK 的 Java 编译器对断言的支持的方式有多种。

要在除 Java 类库之外的其他所有类中启用断言，可使用参数 `-ea`，如下例所示：

```
java -ea PriceChecker
```

要启用一个类中的断言，请在 `-ea` 后加上冒号和类名，如下所示：

```
java -ea:PriceChecker PriceChecker
```

也可以启用特定包中的断言，方法是在 `-ea:` 后加上包名（对于默认包，为...）。

### 提示

标记 `-esa` 用于启用 Java 类库中的断言。您没有理由这样做，因为您不可能去测试这些代码的可靠性。

运行包含断言的类时，如果没有使用标记 `-ea` 或 `esa`，将忽略所有的 `assert` 语句。

由于 Java 新增了关键字 `assert`，因此在程序中，您不能将 `assert` 用作变量名，即使编译程序时没有启用对断言的支持。

接下来的项目 (`CalorieCounter`) 是一个计算器应用程序，它使用了断言。程序清单 7.2 列出了其源代码。

程序清单 7.2 CalorieCounter.java 的完整源代码

```

1: public class CalorieCounter {
2:     float count;
3:
4:     public CalorieCounter(float calories, float fat, float fiber) {
5:         if (fiber > 4) {
6:             fiber = 4;
7:         }
8:         count = (calories / 50) + (fat / 12) - (fiber / 5);
9:         assert count > 0 : "Adjusted calories < 0";
10:    }
11:
12:    public static void main(String[] arguments) {
13:        if (arguments.length < 2) {
14:            System.out.println("Usage: java CalorieCounter calories fat
fiber");
15:            System.exit(-1);
16:        }
17:        try {
18:            int calories = Integer.parseInt(arguments[0]);
19:            int fat = Integer.parseInt(arguments[1]);
20:            int fiber = Integer.parseInt(arguments[2]);
21:            CalorieCounter diet = new CalorieCounter(calories, fat, fiber);
22:            System.out.println("Adjusted calories: " + diet.count);
23:        } catch (NumberFormatException nfe) {
24:            System.out.println("All arguments must be numeric.");
25:            System.exit(-1);
26:        }
27:    }
28: }

```

应用程序 CalorieCounter 根据输入的脂肪量、纤维量和热量计算食品调整后的卡路里总量。这样的程序在体重控制项目中很常见，用于帮助减肥者监控每天的食品摄入量。

这个应用程序接收 3 个命令行参数：calories、fat 和 fiber，它们是作为字符串输入的，第 18~20 行将它们转换为整型值。

构造函数 CalorieCounter 接受 3 个参数，并在第 8 行将它们插入到一个公式中，以计算调整后的卡路里量。

构造函数作出的假设之一是，调整后的量总是为正。这是使用下述 assert 语句来检测的：

```
assert count > 0 : "Adjusted calories < 0";
```

编译这个类后，运行它时应使用标记 -ea，以便断言能够发挥作用，如下例所示：

```
java -ea CalorieCounter 150 3 0
```

上述参数值将导致调整后的卡路里量为 3.25。要查看断言为假的情况，可将卡路里量 30、脂肪 0 克和纤维 6 克作为参数。

断言是 Java 语言中一种不同寻常的特性，因为在大多数情况下，它们都不会导致任何事情发生。它们是一种表达方式，指出类正在正确地运行（以及当类运行时，应为真的事情）。在类中使用断言，可使类更可靠或让您获悉某种假设是错误的，这本身就是很有用的信息。

#### 警告

有些 Java 程序员认为，由于在运行时可以关闭断言，因此它们并非一种可靠的、提高类的可靠性的途径。

## 7.7 线程

使用 Java 进行编程时，需要考虑的问题之一是如何使用系统资源。图形、复杂的数学计算和其他密集型任务可能占用大量的处理器时间。

使用图形用户界面（这是一种软件风格，将在下一周的章节中介绍）的程序尤其如此。

如果您编写的图形 Java 程序需要完成某种占用大量计算机时间的任务，将发现该程序的图形用户界面的响应速度非常慢——下拉式列表需要几秒甚至更长才出现，单击按钮时很久才会响应等。

为解决这种问题，可将 Java 程序中占用大量处理器时间的函数分离出来，让它们同程序的其他部分分开运行。

这可以通过使用 Java 语言中名为线程的特性来实现。

线程是程序的一部分，它们在程序的其他代码执行其他工作时独立运行。这也被称为多任务，因为程序可以同时处理多项任务。

线程适合用于完成占用大量处理时间且连续运行的任务。

通过将程序的工作负荷放到线程中，可以让程序的其他部分处理其他任务。这也使虚拟机能够更容易地对程序进行处理，因为所有的处理器密集型工作都被放到独立的线程中。

### 7.7.1 编写线程化程序

在 Java 中，线程是用包 `java.lang` 中的 `Thread` 类实现的。

最简单的线程用法是，让程序暂停一段时间，并在此期间处于空闲状态。为此，可以调用 `Thread` 类的 `sleep(long)` 方法，并将要暂停的时间（单位为毫秒）作为参数传递给它。

如果被暂停的线程由于某种原因而被中断，该方法将引发 `InterruptedException` 异常（一种可能的原因是，在该线程处于休眠状态时，用户关闭了程序）。

下面的语句将程序暂停 3 秒：

```
try {
    Thread.sleep(3000);
} catch (InterruptedException ie) {
    // do nothing
}
```

`catch` 块没有执行任何操作，您使用 `sleep()` 时，通常都这样做。

线程的用途之一是，将所有耗时的行为都放到单独的类中。

创建线程的方式有两种：创建一个从 `Thread` 派生而来的类或者在另一个类中实现 `Runnable` 接口。

`Thread` 类和 `Runnable` 接口都位于 `java.lang` 包中，因此无需使用任何 `import` 语句就能够引用它们。

由于 `Thread` 类实现了接口 `Runnable`，因此以这两种方法创建的对象将以相同的方式启动和终止线程。

要实现 `Runnable` 接口，将关键字 `implements` 加入到类声明中，并在后面加上接口的名称，如下例所示：

```
public class StockTicker implements Runnable {
    public void run() {
        // ...
    }
}
```

当类实现接口时，它必须包含该接口的所有方法。`Runnable` 接口只包含了一个方法：`run()`。

创建线程的第一步是创建一个到 `Thread` 对象的引用：

```
Thread runner;
```

上述语句创建一个线程引用，但还没有将 `Thread` 对象赋给它。要创建线程，可调用构造函数 `Thread(Object)`，并将要线程化的对象作为参数传递给它。用下面的语句创建了一个线程化的 `StockTicker` 对象：

```
StockTicker tix = new StockTicker();
Thread tickerThread = new Thread(tix);
```

适合创建线程的两个地方是：应用程序的构造函数和组件（如面板）的构造函数。

要启动线程，可调用它的 `start()` 方法，如下面的语句所示：

```
tickerThread.start();
```

下面的语句可用在线程类中启动线程：

```
Thread runner;
if (runner == null) {
    runner = new Thread(this);
    runner.start();
}
```

构造函数 `Thread()` 中的关键词 `this` 指的是包含这些语句的对象。将对象赋给变量 `runner` 之前，它的值为 `null`，因此 `if` 语句确保线程不会被多次启动。

要运行线程，可调用其 `start()` 方法，如前一个例子中的下述语句所示：

```
runner.start();
```

调用线程的 `start()` 方法将导致另一个方法，即方法 `run()` 被调用。线程化对象中必须包含 `run()` 方法。

`run()` 方法是线程类的引擎。前面介绍线程时指出过，线程是一种将处理器密集型工作隔离，使之独立于类中其他部分运行的途径。这些行为将包含在线程的 `run()` 方法及其调用的方法中。

## 7.7.2 线程化应用程序

线程化程序要求在不同对象之间进行大量的交互，下面通过一个示例来更清晰地说明这一点。

程序清单 7.3 的类查找素数，如第 10 个素数、第 100 个素数、第 1 000 个素数。这需要一些时间，尤其是当要检查的值超过 100 000 时，因此将查找素数的代码放在一个独立的线程中。

请在 Java 编辑器中输入程序清单 7.3 中的代码，并将其保存为 `PrimerFinder.java`。

程序清单 7.3 完整的 `PrimeFinder.java` 源代码

```
1: public class PrimeFinder implements Runnable {
2:     public long target;
3:     public long prime;
4:     public boolean finished = false;
5:     private Thread runner;
6:
7:     PrimeFinder(long inTarget) {
8:         target = inTarget;
9:         if (runner == null) {
10:            runner = new Thread(this);
11:            runner.start();
12:        }
13:    }
14:
15:    public void run() {
16:        long numPrimes = 0;
17:        long candidate = 2;
18:        while (numPrimes < target) {
19:            if (isPrime(candidate)) {
20:                numPrimes++;
21:                prime = candidate;
22:            }
23:            candidate++;
24:        }
25:        finished = true;
26:    }
27:
28:    boolean isPrime(long checkNumber) {
29:        double root = Math.sqrt(checkNumber);
30:        for (int i = 2; i <= root; i++) {
31:            if (checkNumber % i == 0)
32:                return false;
33:        }
34:        return true;
35:    }
36: }
```



编译 PrimeFinder 类。这个类没有 main() 方法，因此不能将其作为应用程序进行运行。接下来将创建一个使用这个类的程序。

PrimeFinder 类实现了 Runnable 接口，因此可将其作为线程运行。

其中有 3 个公有的实例变量。

- target: 一个 Long 变量，用于指出要查找的几个素数。如果要查找第 5 000 个素数，则 target 的值为 5 000。

- prime: 一个 Long 变量，用于存储找到的最后一个素数。

- finished: 一个布尔变量，指出是否达到目标。

还有一个名为 runner 的私有实例变量，用于存储一个线程对象，这个类将在该线程中运行。线程启动之前，该变量应为 null。

第 7~13 行的构造函数 PrimeFinder 设置 target 的值，并启动线程（如果还没有启动的话）。当线程的 start() 方法被调用时，它将调用线程化类的 run() 方法。

run() 方法位于第 15~26 行。线程（通常是一个线程化类）的大部分工作是在该方法中完成的。您希望将计算密集型任务放到独立的线程中，以免使程序的其他部分陷入停顿。

这个方法使用了两个新的变量：numPrimes 和 candidate，前者指出已找到多少个素数，后者可能是素数的数字。candidate 的初值被设置为第一个可能的素数——2。

第 18~24 行的循环将不断执行，直到找到指定数目的素数。

首先，它调用 isPrime(long) 方法，检查当前的 candidate 值是否为素数。如果是，该方法返回 true，否则返回 false。

如果 candidate 是素数，则将 numPrimes 的值加 1，并将该素数赋给实例变量 prime。

然后将 candidate 的值加 1，并进入下一次循环。

找到要找的素数后，while 循环将结束，而 finished 变量将被设置为 true，这表明 PrimeFinder 对象已经找到所需的素数，并结束搜索。

到达第 26 行后，run() 方法将结束，线程不再做任何工作。

isPrime() 方法位于第 28~35 行，它使用运算符 % 来判断一个数是否为素数，该运算符返回除法运算的余数。如果一个数能被 2 或更大的数整除（余数为 0），则说明它不是素数。

程序清单 7.4 是一个使用 PrimeFinder 类的应用程序，请输入其中的代码，并将其保存为文件 PrimeThreads.java。

程序清单 7.4 完整的 PrimeThreads.java 源代码

```

1: public class PrimeThreads {
2:     public static void main(String[] arguments) {
3:         PrimeThreads pt = new PrimeThreads(arguments);
4:     }
5:
6:     public PrimeThreads(String[] arguments) {
7:         PrimeFinder[] finder = new PrimeFinder[arguments.length];
8:         for (int i = 0; i < arguments.length; i++) {
9:             try {
10:                 long count = Long.parseLong(arguments[i]);
11:                 finder[i] = new PrimeFinder(count);
12:                 System.out.println("Looking for prime " + count);
13:             } catch (NumberFormatException nfe) {
14:                 System.out.println("Error: " + nfe.getMessage());
15:             }
16:         }
17:         boolean complete = false;
18:         while (!complete) {
19:             complete = true;
20:             for (int j = 0; j < finder.length; j++) {
21:                 if (finder[j] == null) continue;
22:                 if (!finder[j].finished) {

```



```

23:         complete = false;
24:     } else {
25:         displayResult(finder[j]);
26:         finder[j] = null;
27:     }
28: }
29: try {
30:     Thread.sleep(1000);
31: } catch (InterruptedException ie) {
32:     // do nothing
33: }
34: }
35: }
36:
37: private void displayResult(PrimeFinder finder) {
38:     System.out.println("Prime " + finder.target
39:         + " is " + finder.prime);
40: }
41: }

```

输入上述代码后，保存并对其进行编译。

应用程序 PrimeThreads 可用于查找一个或多个素数。用户可通过命令行参数指定要查找第几个素数，且可以查找任何数目的素数。

如果您使用的是 JDK，下面的示例演示了如何使用该应用程序：

```
java PrimeThreads 1 10 100 1000
```

其输出如下：

```

Looking for prime 1
Looking for prime 10
Looking for prime 100
Looking for prime 1000
Prime 1 is 2
Prime 10 is 29
Prime 100 is 541
Prime 1000 is 7919

```

应用程序 PrimeThreads 的第 8~16 行中的 for 循环将为每一个命令行参数创建一个 PrimeFinder 对象。

由于参数是 String，而构造函数 PrimeFinder 需要一个 long 参数，因此使用类方法 Long.parse(String) 来处理这种转换。由于所有的数字-分析方法都可能引发 NumberFormatException 异常，因此必须将这种方法调用放在 try-catch 块中，以应付参数不能转换为数字的情况。

PrimeFinder 对象被创建后，它便开始在自己的线程中运行（这是由 PrimeFinder 构造函数指定的）。

第 18~34 行的 while 循环检查所有的 PrimeFinder 线程是否都已结束。这是通过检查实例变量 finished 是否为 true 来实现的。线程结束后，第 25 行调用方法 displayResult() 来显示找到的素数，然后将线程设置为 null，为无用单元收集释放对象（并防止其结果被显示多次）。

第 30 行的方法调用 Thread.sleep(1000) 导致 while 循环每结束一次迭代后都暂停 1 秒。暂停循环可避免因 Java 解释器不断地执行循环中的语句而陷入困境。

### 7.7.3 终止线程

终止线程比启动线程要复杂些。类 Thread 包含一个 stop() 方法，可以调用它来终止线程，但它可能导致 Java 运行环境不稳定，并可能在程序中引入难于检测的错误。因此，该方法被摒弃了，也就是说，在有其他技术可用时，不应使用该方法。

一种更好的线程终止方式是，让线程的 run() 方法的循环在某个变量的值发生变化后结束，如下例所示：

```

public void run() {
    while (okToRun == true) {

```

```

    }
    // ...
}

```

变量 `okToRun` 可以是线程类的实例变量，如果它变成 `false`，则 `run()` 方法中的循环将结束。

还有一种终止线程的方式是，仅当当前运行的线程有一个指向它的变量时，才执行方法 `run()` 中的循环。

在前面的例子中，一个名为 `runner` 的 `Thread` 对象被用来存储当前的线程。

类方法 `Thread.currentThread()` 返回指向当前线程（即对象在其中运行的线程）的引用。

只要 `runner` 和 `currentThread()` 指向相同的对象，下面的方法 `run()` 就将继续循环：

```

public void run() {
    Thread thisThread = Thread.currentThread();
    while (runner == thisThread) {
        // ...
    }
}

```

如果您使用了类似这样的循环，则可以在类的任何地方使用下面的语句来终止线程：

```
runner = null;
```

## 7.8 总结

异常、断言和线程有助于优化程序设计和增强其健壮性。

异常让您能够管理程序中潜在的错误。通过使用 `try`、`catch` 和 `finally`，可以保护可能导致异常的代码，在异常发生时处理它们。

处理异常仅是问题的一半，另一半是生成和引发异常。`throws` 子句告诉方法的用户，该方法可能引发异常；它也可用于在方法体的方法调用中将异常传递出去。

您学习了如何创建并引发异常：定义新的异常类并使用 `throw` 引发异常类的实例。

断言让您能够使用条件语句和布尔值来指出程序运行正常。如果程序运行不正常，将引发断言异常。

线程让您能够将类中处理器-密集型部分与其他部分分开运行，当类需要执行计算密集型任务（如动画、复杂的数学运算或遍历大量数据）时，这很有用。

您还可以使用线程来同时执行多项任务，并从外部启动和终止线程。

线程实现了 `Runnable` 接口，后者包含一个方法：`run()`。通过调用线程的 `start()` 方法来启动线程时，线程的 `run()` 方法将自动被调用。

## 7.9 问与答

问：我还是不太确信自己理解了异常、错误和运行阶段异常之间的区别。有其他的方式来看待它们吗？

答：错误是由动态链接或虚拟机问题引起的，因而大多数程序员无需关心它们，即使关心也无法处理。

运行阶段异常是由 Java 代码的正常执行引起的，虽然它们偶尔反映了应显式处理的情况，但通常反映的是编码错误，因而只需打印出错误消息来帮助标记该错误。

非运行阶段异常（如 `IOException`）必须通过健壮的、深思熟虑的代码来显式地处理。编写 Java 类库时，只使用了其中的几个，但它们对于确保安全、正确地使用系统至关重要。编译器通过其 `throws` 子句的检测和限制来帮助您正确地处理这些异常。

问：在使 Java 程序更可靠方面，断言和单元测试之间有何异同？

答：和断言一样，单元测试是一种通过添加测试确保软件可靠的方法。JUnit 是一个开源库，可从网站 <http://www.junit.org> 下载，它是 Java 程序员中最流行的单元测试框架。

使用 JUnit 可编写一组测试，它们创建您开发的 Java 对象并调用其方法。然后检查这些测试生成的值是否与期望的相同。必须通过所有测试，软件才被视为通过了测试。

虽然单元测试不会比您创建的测试好，但修改软件时，现有的测试集很有帮助。通过在修改软件后运行测试，可让您更确信软件能够正确运行。

大多数 Java 程序员都使用单元测试而不是断言，有些 Java 程序员甚至在编写代码前就编写测试。

问：有办法避开 throws 子句对方法所做的严格限制吗？

答：有。假设您经过仔细考虑，决定需要避开这种限制。这种情况几乎不会发生，因为正确的解决办法是，重新对方法进行设计，使之反映需要引发的异常。然而，假设是由于系统类让您陷入困境，则首选的解决方案是，通过继承 RuntimeException 来创建一个无需检查的新异常。这样，您可以按自己的意愿来引发异常，因为一直困扰着您的 throws 子句并不需要包含新的异常。如果需要很多这样的异常，则妥善的解决之道是，在新的 Runtime 类中实现一些新的异常接口。您可以自由地选择捕捉这些新接口的一部分（常规的 Runtime 异常都无需捕获），而忽略其余的 Runtime 异常（如果不这样做，将需要通过库中令人讨厌的标准方法来处理）。

## 7.10 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 7.10.1 问题

1. 可以使用哪个关键词来跳出 try 块，并进入 finally 块？
  - a. catch;
  - b. return;
  - c. while。
2. 在 Java 中创建异常时，必须以哪个类为超类？
  - a. Throwable;
  - b. Error;
  - c. Exception。
3. 如果一个类实现了 Runnable 接口，则这个类必须包含哪些方法？
  - a. start()、stop() 和 run();
  - b. actionPerformed();
  - c. run()。

答案

1. b。
2. c, Throwable 和 Error 主要供 Java 本身使用，在程序中需要注意的错误属于 Exception 层次结构。
3. c, Runnable 接口只要求实现 run() 方法。



### 7.10.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

应用程序 `AverageValue` 从命令行接受 10 个浮点数参数，并显示它们的平均值。

对于下述代码：

```
public class AverageValue {
    public static void main(String[] arguments) {
        float[] temps = new float[10];
        float sum = 0;
        int count = 0;
        int i;
        for (i = 0; i < arguments.length & i < 10; i++) {
            try {
                temps[i] = Float.parseFloat(arguments[i]);
                count++;
            } catch (NumberFormatException nfe) {
                System.out.println("Invalid input: " + arguments[i]);
            }
            sum += temps[i];
        }
        System.out.println("Average: " + (sum / i));
    }
}
```

哪条语句有错？

- a. `for (i = 0; i < arguments.length & i < 10; i++) {`
- b. `sum += temps[i]`
- c. `System.out.println("Average: " + (sum / i))`
- d. 没有，程序是正确的

答案 c

### 7.11 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 修改应用程序 `PrimeFinder` 类，使之在传递给构造函数的参数为负数时，引发一个新的异常 `NegativeNumberException`。
2. 修改应用程序 `PrimeThreads`，使之能够处理新的 `NegativeNumberException` 错误。





## 第 2 周课程

### Java 类库

第 8 章 数据结构

第 9 章 使用 Swing

第 10 章 创建 Swing 界面

第 11 章 在用户界面上排列组件

第 12 章 响应用户输入

第 13 章 使用颜色、字体和图形

第 14 章 开发 Swing 应用程序

## 第 8 章

# 数据结构

第 1 周的课程介绍了 Java 语言中的核心元素：对象、类、继承以及关键字、语句、表达式和运算符。

第 2 周的课程将重点从您创建类转到已经为您创建好的类：Java 类库是 Sun Microsystems 公司提供的一套标准包，其中包含 1 000 多个类，您可以在自己的 Java 程序中使用它们。

本章的课程将首先介绍可用来表示数据的类。

### 8.1 超越数组

在 Java 类库的 `java.util` 包中有一组数据结构，它们让您能够更灵活地组织和操纵数据。

深入理解数据结构及何时使用它们将对 Java 编程工作有极大的帮助。

您创建的很多 Java 程序都依赖于某种方式来存储和操纵类中的数据。到现在为止，您使用了 3 种数据结构来存储和检索数据：变量、String 对象和数组。

这些只是 Java 提供的数据类型中的很少一部分。如果不了解编程中可用的各种数据结构，您将在使用其他数据结构效率更高，也更容易实现时，却使用数组或字符串。

除基本数据类型和字符串外，数组是 Java 支持的最简单的数据结构。数组是一系列的数据元素，它们都属于某种基本类型或类。数组被视为单个对象，就像基本数据类型变量一样，但包含多个可被独立存取的元素。需要存储和访问相关信息时，数组很有用。

数组最大的缺点在于，不能改变其大小以存储更多或更少的元素。这意味着您不能向已满的数组中加入新的元素。本章将介绍的链表和矢量没有这种限制。

#### 注意

与 `java.util` 包提供的数据结构不同，数组是 Java 的核心元素，因此是使用 Java 实现的。因此，您可以在 Java 中使用数组而不用导入任何包。

### 8.2 Java 数据结构

`java.util` 包提供的数据结构的功能非常强大，具备众多的功能。这些数据结构包括接口 `Iterator`、`Map` 以及下述类：

- `BitSet`;
- `Vector`;
- `Stack`;
- `Hashtable`。

这些数据结构都提供了一种以明确的方式存储和检索信息的方式。接口 `Iterator` 本身并非数据结构，但它定义了一种连续地检索数据结构中的元素的方式。例如，`Iterator` 定义了一个 `next()` 方法，该方法取得包含多个元素的数据结构中的下一个元素。

**注意**

Iterator 是 Enumeration 接口的扩展和改进版本。虽然 Enumeration 仍被支持, 但 Iterator 的方法名更简单, 且支持删除元素。

Bitset 类实现了一组位或标记 (flag), 这些位可被分别设置或清除。当需要跟踪一组布尔值时, 这种类型很有用。您只需让每一位对应一个值, 并根据需要设置或清除即可。

标记 (flag) 是一个布尔值, 表示程序中的一组开/关状态之一。

Vector 类似于传统的 Java 数组, 只是它可以根据需要增大 (以存储新元素) 和缩小。与数组一样, Vector 对象的元素也可以通过索引来访问。Vector 类的优点是, 无需在创建时将它设置成特定的大小, 必要时它将自动地增大或缩小。

Stack 类实现了一个后进先出的元素堆栈。可以将 Stack 看作一个垂直的对象堆栈, 新元素被加入到栈顶。从栈中弹出元素时, 弹出的是栈顶元素。换句话说, 最后加入的元素将首先被弹出。被弹出的元素将从堆栈中删除, 这不像数组, 数组中的元素总是可用的。

Hashtable 实现了 Dictionary, 后者是一个抽象类, 它定义了一种用于将键值与数值关联起来的数据结构。当需要通过键值而不是整数索引来访问数据时, 这个类很有用。由于 Dictionary 类是抽象的, 所以它只提供了键映射数据结构的框架, 而没有给出具体的实现。

键值 (key) 是一个标识符, 用于引用或查找数据结构中的值。

键映射数据结构的实现是由类 Hashtable 提供的, 它根据用户定义的键值结构来组织数据。例如, 在存储在散列表中的邮政编码列表中, 可以将编码作为键值来存储和排序数据。在散列表中, 键值的具体含义取决于散列表的用法及其包含的数据。

接下来的几节将更详细地介绍这些数据结构, 以说明它们的工作原理。

### 8.2.1 Iterator

接口 Iterator 提供了一种以定义好的顺序遍历一系列元素 (这是很多数据结构都需要完成的任务) 的标准方式。虽然不能在数据结构外使用这个接口, 但了解 Iterator 接口的工作原理将有助于您理解其他 Java 数据结构。

下面来看看由接口 Iterator 定义的一些方法:

```
public boolean hasNext() {
    // body of method
}

public Object next() {
    // body of method
}

public void remove() {
    // body of method
}
```

方法 hasNext() 定义了结构是否还包含其他元素。您调用该方法来查看是否可以继续遍历结构。

方法 next() 获得结构中的下一个元素。如果没有更多的元素, next() 将引发 NoSuchElementException 异常。为避免产生这种异常, 应结合使用 hasNext() 和 next() 来确保还有元素可检索。

下面的 while 循环使用这两种方法来遍历一个名为 users 的数据结构对象, 该对象实现了接口 Iterator:

```
while (users.hasNext()) {
    Object ob = users.next();
    System.out.println(ob);
}
```

上述代码使用 hasNext() 和 next() 方法显示了每个列表项的内容。

方法 next() 总是返回一个 Object 对象, 您可以将其转换为数据结构存储的类对象。例如, 下面的

例子将其转换为 String 对象:

```
while (users.hasNext()) {
    String ob = (String) users.next();
    System.out.println(ob);
}
```

**注意**

由于 Iterator 是接口, 所以不能直接将它用作数据结构。相反, 您将在其他数据结构中使用 Iterator 定义的方法。这种体系结构的意义在于, 它为很多标准的数据结构提供了一致的接口, 这使得它们学习和使用起来更容易。

### 8.2.2 位组

需要表示大量的二进制数据 (即只可以为 1 或 0 的比特值) 时, BitSet 类很有用。这些值也被称为开/关值 (1 表示开, 0 表示关) 或布尔值 (1 表示真, 0 表示假)。

使用 BitSet 类, 可以用位来存储布尔值, 而无需通过按位运算来提取位值。您只需使用索引来引用每一位。另一个优点是, 它可以自动增大, 以表示程序所需的位数。图 8.1 显示了位组数据结构的逻辑组织。

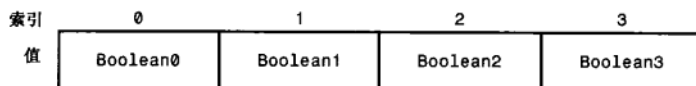


图 8.1 位组数据结构的组织

可以使用 BitSet 来存储可以用布尔值来模拟的属性。由于位组中的各个位是通过索引来访问的, 所以可以将每个属性名定义为常量索引值, 如下面的类所示:

```
class ConnectionAttributes {
    public static final int READABLE = 0;
    public static final int WRITABLE = 1;
    public static final int STREAMABLE = 2;
    public static final int FLEXIBLE = 3;
}
```

在上述类中, 属性名对应的值从 0 开始依次递增。您可以使用这些值来取得或设置位组中的位。但首先必须创建一个 BitSet 对象:

```
BitSet connex = new BitSet();
```

该构造函数创建了一个大小没有被指定的位组。您还可以创建一个指定大小的位组:

```
BitSet connex = new BitSet(4);
```

这创建了一个包含 4 个布尔位的位组。不管使用哪个构造函数, 新位组中的所有位都将被初始化为 false。创建位组后, 便可以使用 set(int) 和 clear(int) 方法以及您定义的索引常量来设置和清除这些位:

```
connex.set(ChannelAttributes.WRITABLE);
connex.set(ChannelAttributes.STREAMABLE);
connex.set(ChannelAttributes.FLEXIBLE);
```

```
connex.clear(ChannelAttributes.WRITABLE);
```

上述代码设置了属性 WRITABLE、STREAMABLE 和 FLEXIBLE, 然后清除 WRITABLE 属性。对于每个属性, 还使用了类名, 这是因为这些常量都是类 ChannelAttributes 中的类变量。

可以使用 get 方法来获得位组中的各个位:

```
boolean isWritable = connex.get(ChannelAttributes.WRITABLE);
```

使用 size 方法, 可以确定位组表示了多少位:

```
int numBits = connex.size();
```

类 BitSet 还可以提供用于对位组进行比较以及诸如 AND、OR 和 XOR 等按位运算的方法。这些方法都接受一个 BitSet 对象作为其唯一的参数。

本章的第一个工程 `HolidaySked` 是一个 Java 类，它使用位组来跟踪一年中的哪些天是节假日。由于 `HolidaySked` 必须能够判断任何一天是否为节假日，因此使用了位组。请在编辑器中输入程序清单 8.1 中的代码，然后将其保存为文件 `HolidaySked.java`。

程序清单 8.1 完整的 `HolidaySked.java` 源代码

```

1: import java.util.*;
2:
3: public class HolidaySked {
4:     BitSet sked;
5:
6:     public HolidaySked() {
7:         sked = new BitSet(365);
8:         int[] holiday = { 1, 15, 50, 148, 185, 246,
9:             281, 316, 326, 359 };
10:        for (int i = 0; i < holiday.length; i++) {
11:            addHoliday(holiday[i]);
12:        }
13:    }
14:
15:    public void addHoliday(int dayToAdd) {
16:        sked.set(dayToAdd);
17:    }
18:
19:    public boolean isHoliday(int dayToCheck) {
20:        boolean result = sked.get(dayToCheck);
21:        return result;
22:    }
23:
24:    public static void main(String[] arguments) {
25:        HolidaySked cal = new HolidaySked();
26:        if (arguments.length > 0) {
27:            try {
28:                int whichDay = Integer.parseInt(arguments[0]);
29:                if (cal.isHoliday(whichDay)) {
30:                    System.out.println("Day number " + whichDay +
31:                        " is a holiday.");
32:                } else {
33:                    System.out.println("Day number " + whichDay +
34:                        " is not a holiday.");
35:                }
36:            } catch (NumberFormatException nfe) {
37:                System.out.println("Error: " + nfe.getMessage());
38:            }
39:        }
40:    }
41: }

```

`HolidaySked` 类只包含一个实例变量：`sked`，这是一个位组，将被用来存储一年中的每一天对应的值。

这个类的构造函数方法位于第 6~13 行，它创建位组 `sked`（包含 365 位）。位组被创建时，其中的所有位都被设置为 0。

接下来，创建了一个名为 `holiday` 的 `int` 数组。该数组用于存储 2007 年中各个节假日的编号，其中第一个节假日的编号为 1（元旦），最后一个节假日的编号为 359（圣诞）。

数组 `holiday` 被用来设置位组 `sked` 中对应于节假日的位。第 10~12 行的 `for` 循环遍历 `holiday` 数组，对于其中的每个元素，都调用方法 `addHoliday(int)`。

方法 `addHoliday(int)` 位于第 15~17 行。其中的参数表示这天是节假日。为将指定的位设置为 1，调用了位组的 `set(int)` 方法。例如，调用 `set(359)` 将把第 359 位的值设置为 1。

`HolidaySked` 类还能够判断某天是否为节假日。这是通过第 19~22 行的 `isHoliday(int)` 方法实现的。该方法调用位组的 `get(int)` 方法，如果指定的位的值为 1，则 `get()` 方法返回 `true`，否则返回 `false`。

这个类可作为应用程序运行，因为它包含一个 `main()` 方法（第 24~40 行）。该应用程序接受一个

命令行参数：一个 1~365 的数字，表示一年中的某一天。该应用程序根据 Holidaysked 类的一览表指出这一天是否为节假日。请使用诸如 15（马丁路德金纪念日）或 103（作者的 40 岁生日）等值来测试该应用程序。该应用程序将指出第 15 天是节假日（美国），但第 103 天不是。

### 8.2.3 Vector

在本章介绍的数据结构中，类 Vector 可能是最流行的，它实现了矢量——一个可缩放的对象数组。由于类 Vector 负责根据需要改变大小，所以它必须根据元素的增加和删除决定缩放多少。您可以在创建时很容易地控制矢量的这个方面。

介绍这方面的内容之前，先来看一下如何创建基本矢量：

```
Vector v = new Vector();
```

该构造函数创建了一个不含任何元素的默认矢量。实际上，所有的矢量在创建时都为空。判断矢量大小的属性之一是其初始容量或默认为多少个元素分配了内存。

矢量的大小（size）指的是它当前存储的元素数目。

矢量的容量（capacity）是被分配来存储元素的内存量，它总是大于或等于矢量的大小。

下面的代码演示了如何创建指定容量的矢量：

```
Vector v = new Vector(25);
```

该矢量将被分配足够的内存，以支持 25 个元素。然而加入 25 个元素后，矢量必须判断如何进行扩展以接受更多元素。可以使用另一个 Vector 构造函数来指定矢量每次的增长量。

```
Vector v = new Vector(25, 5);
```

该矢量的初始长度为 25 个元素，在加入 25 个元素后，它将以每次 5 个元素的速度增长。这意味着该矢量的大小将依次增加到 30、35 等。增长值越小，内存管理的效率越高，但执行开销越大，因为执行内存分配的次数将越多，增长值越大，执行内存分配的次数将越少，但如果没有用完分配的所有空间，将浪费内存。

不能像在数组那样使用方括号（[ ]）来访问矢量中的元素，而必须使用 Vector 类中定义的方法来访问。add() 方法用于将元素加入到矢量中，如下例所示：

```
v.add("Pak");
v.add("Han");
v.add("Inkster");
```

上述代码演示了如何将字符串加入到矢量中。要检索最后加入的字符串，可以使用方法 lastElement()：

```
String s = (String)v.lastElement();
```

注意，必须对 lastElement() 的返回值进行强制类型转换，因为类 Vector 是被设计成处理 Object 类的。

方法 get() 让您能够通过索引来检索矢量中的元素，如下例所示：

```
String s1 = (String)v.get(0);
String s2 = (String)v.get(2);
```

由于矢量的索引值是从 0 开始的，所以第一个 get() 调用返回的是字符串“Pak”，而第二个调用返回字符串“Han”。正如可以检索特定位置的元素一样，您也可以使用方法 add() 和 remove() 来将元素加入到指定位置和删除指定位置的元素：

```
v.add(1, "Park");
v.add(0, "Sorenstam");
v.remove(3);
```

第一个 add() 调用将一个元素加入到第二个位置，即“Pak”和“Han”之间。为容纳插入的字符串“Park”，字符串“Han”和“Inkster”被向后移动一个位置。第二个 add() 调用将一个元素插入到第一个位置——矢量的开头。原来的元素都被向后移动一个位置，以容纳插入的字符串“Sorenstam”。现

在，这个矢量的内容如下：

- “Sorenstam”;
- “Pak”;
- “Park”;
- “Han”;
- “Inkster”。

`remove()`调用删除索引值为3的元素，即字符串“Han”。这样，该矢量将包含如下字符串：

- “Sorenstam”;
- “Pak”;
- “Park”;
- “Inkster”。

可以使用 `set()`方法来修改指定的元素：

```
v.set(1, "Kung");
```

上述代码将字符串“Pak”替换为字符串“Kung”，最后得到的矢量如下：

- “Sorenstam”;
- “Kung”;
- “Park”;
- “Inkster”。

要清除矢量中的所有内容，可以使用方法 `clear()`来删除所有的元素：

```
v.clear();
```

类 `Vector` 还提供了一些不通过索引来处理元素的方法。这些方法在矢量中搜索特定的元素，其中第一个是方法 `contains()`，它检验矢量中是否包含指定的元素：

```
boolean isThere = v.contains("Webb");
```

另一个以这种方式工作的方法是 `indexOf()`方法，它找出元素对应的索引：

```
int i = v.indexOf("Inkster");
```

如果矢量中包含指定的元素，`indexOf()`方法将返回它的索引，如果没有，则返回-1。方法 `removeElement()`的工作原理与此类似，它删除指定的元素，而不是指定索引处的元素：

```
v.removeElement("Kung");
```

类 `Vector` 提供了一些用于判断和操纵矢量大小的方法。

首先，方法 `size` 判断矢量中的元素数目：

```
int size = v.size();
```

要显式地设置矢量的大小，可以使用方法 `setSize()`：

```
v.setSize(10);
```

方法 `setSize()`将矢量缩放为指定的大小。如果矢量被扩展，将插入空元素；如果矢量被压缩，索引值超过指定大小的元素都将被丢弃。

矢量有两个与大小相关的不同属性：大小和容量。大小是矢量中的元素数目，而容量是分配用来存储元素的内存量。容量总是大于或等于大小。可以使用方法 `trimToSize()`来使容量与大小相等：

```
v.trimToSize();
```

您还可以使用方法 `capacity()`来查看容量：

```
int capacity = v.capacity();
```

## 8.2.4 遍历数据结构

如果要依次处理矢量中的所有元素，可使用方法 `iterator()`，它返回一个可供您遍历的元素列表：

```
Iterator it = v.iterator();
```



正如本章前面指出的，可以使用迭代器来依次遍历元素。在这个例子中，您可以使用接口 `Iterator` 定义的方法来处理列表 `it`。

下面的 `for` 循环使用迭代器及其方法来遍历整个矢量：

```
for (Iterator i = v.iterator(); i.hasNext(); ) {
    String name = (String) i.next();
    System.out.println(name);
}
```

接下来的项目将演示如何处理矢量。程序清单 8.2 中的 `CodeKeeper` 类存储了一组文本代码，其中一些是由类提供的，其他的是用户提供的。由于在程序执行前，并不知道为存储这些代码需要多少空间，因此使用矢量而不是数组来存储这些数据。

程序清单 8.2 `CodeKeeper.java` 的完整源代码

```
1: import java.util.*;
2:
3: public class CodeKeeper {
4:     Vector list;
5:     String[] codes = { "alpha", "lambda", "gamma", "delta", "zeta" };
6:
7:     public CodeKeeper(String[] userCodes) {
8:         list = new Vector();
9:         // load built-in codes
10:        for (int i = 0; i < codes.length; i++) {
11:            addCode(codes[i]);
12:        }
13:        // load user codes
14:        for (int j = 0; j < userCodes.length; j++) {
15:            addCode(userCodes[j]);
16:        }
17:        // display all codes
18:        for (Iterator ite = list.iterator(); ite.hasNext(); ) {
19:            String output = (String) ite.next();
20:            System.out.println(output);
21:        }
22:    }
23:
24:    private void addCode(String code) {
25:        if (!list.contains(code)) {
26:            list.add(code);
27:        }
28:    }
29:
30:    public static void main(String[] arguments) {
31:        CodeKeeper keeper = new CodeKeeper(arguments);
32:    }
33: }
```

这个类能够通过编译，但将出现警告，指出它使用了未经检查或不安全的操作。这并没有听起来那么严重：代码将正常运行，也并非不安全的。

这种警告给出了一个不那么巧妙的提示：还有更好、更安全的方法来使用矢量和数据结构，本章后面将介绍这种技术。

`CodeKeeper` 类使用一个名为 `list` 的 `Vector` 实例变量来存储文本代码。

首先，从字符串数组中将 5 个内置的编码读取到矢量中（第 10~12 行）。

接下来，加入用户以命令行参数提供的编码（第 14~16 行）。

编码是通过调用方法 `addCode()`（第 24~28 行）加入的。仅当新的文本编码没有出现在矢量中时才会被加入，这种判断是使用矢量的 `contains(Object)` 方法做出的。

将编码加入到矢量中后，将显示其内容。运行这个应用程序，并指定命令行参数“gamma”、“beta”和“delta”时，将得到如下输出：

```
alpha
lambda
```

```

gamma
delta
zeta
beta

```

可使用一种 for 循环来遍历数据结构，这种循环的格式为 `for (variable : structure)`，其中 `structure` 是一个实现了 `Iterator` 的数据结构，而 `variable` 声明了一个对象，用于在循环过程中存储结构中的每个元素。

这种新的 for 循环使用迭代器及其方法来遍历整个矢量：

```

for (Object name : list) {
    System.out.println(name);
}

```

这种新循环可用于任何支持 `Iterator` 接口的数据结构。

### 8.2.5 堆栈

堆栈是一种经典的数据结构，用于模拟以特定顺序进行访问的信息。在 Java 中，`Stack` 类被实现成一个后进先出 (LIFO) 的堆栈，这意味着最后加入的项将首先被弹出。图 8.2 说明了堆栈的逻辑结构。

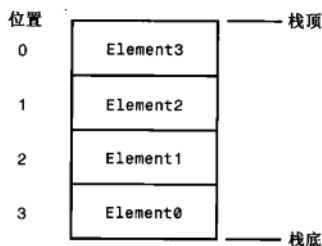


图 8.2 堆栈的组织结构

您可能会问，为什么元素的编号与它们从堆顶算起的位置不一致？别忘了，元素被加入到栈顶，因此位于栈底的 `Element0` 是第一个被加入的元素。同样的，栈顶的 `Element3` 是最后一个被加入的元素。另外，由于 `Element3` 位于栈顶，所以它将首先被弹出。

类 `Stack` 只定义了一个构造函数，这是一个默认构造函数，它创建一个空栈。使用这个构造函数来创建堆栈的方式如下：

```
Stack s = new Stack();
```

在 Java 中，堆栈是 `Vector` 的子类，因此可以像处理其他矢量那样处理堆栈。堆栈还包含了专门用于操纵堆栈的方法。

您使用 `push()` 方法将新的元素加入到堆栈中，这将把元素压入到栈顶：

```

s.push("One");
s.push("Two");
s.push("Three");
s.push("Four");
s.push("Five");
s.push("Six");

```

上述代码将 6 个字符串压入到堆栈中，最后一个字符串（“Six”）位于栈顶。

可使用 `pop()` 方法将元素从堆栈中弹出：

```

String s1 = (String)s.pop();
String s2 = (String)s.pop();

```

上述代码将最后两个字符串从堆栈中弹出，保留前 4 个字符串。上述代码导致变量 `s1` 中包含字符串 “Six”，变量 `s2` 中包含字符串 “Five”。

如果要获得栈顶元素，而并不将它从栈中弹出，可使用 `peek()`：

```
String s3 = (String)s.peek();
```

上述 `peek()` 调用返回字符串 “Four”，但该字符串仍留在堆栈中。可以使用方法 `search()` 在堆栈中搜索元素：

```
int i = s.search("Two");
```

如果找到，方法 `search()` 返回该元素离栈顶的距离；如果没找到，则返回 -1。在上述范例中，字符串 “Two” 是从栈顶算起的第 3 个元素，因此方法 `search()` 返回 2（第一个元素为 0）。

#### 注意

与涉及索引和列表的 Java 数据结构一样，类 `Stack` 在报告元素位置时，也以 0 开始。这意味着栈顶元素的位置为 0，第 4 个元素的位置为 3。

在类 `Stack` 中定义的最后一个方法是 `empty`，用来判断堆栈是否为空：

```
boolean isEmpty = s.empty();
```

虽然 `Stack` 类的用途不如 `Vector` 类那么大，但它提供了一种软件开发中常用的数据结构的功能。

### 8.2.6 Map

接口 `Map` 为实现键映射数据结构定义了一个框架，这种结构可用于存储通过键值引用的对象。键值的作用与数组中的索引相同，它是一个独一无二的值，可用来存取数据结构中指定位置的数据。

您可以使用类 `Hashtable` 或其他实现了接口 `Map` 的类来实现键映射方法。类 `Hashtable` 将在下一节介绍。

接口 `Map` 定义了一种根据键值来存储和检索信息的方式。这与类 `Vector` 类似，在 `Vector` 中，元素是通过索引（一种特殊的键值）来存取的。然而，在接口 `Map` 中，键值可以是任何东西。您可以创建自己的类，并将其作为键值来存取和操纵字典中的数据。图 8.3 说明了在字典中，键值是如何映射到数据的。

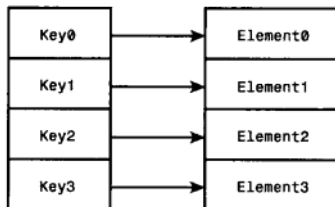


图 8.3 键映射数据结构的组织结构

接口 `Map` 声明了大量用于操纵字典中的数据的方法。实现类必须实现所有的这些方法才能真正有用。方法 `put` 和 `get` 分别用于将对象放到字典中和取得字典中的对象。

假设 `look` 是一个实现了 `Map` 接口的类，下面的代码演示了如何使用方法 `put` 来加入元素：

```
Rectangle r1 = new Rectangle(0, 0, 5, 5);
look.put("small", r1);
Rectangle r2 = new Rectangle(0, 0, 15, 15);
look.put("medium", r2);
Rectangle r3 = new Rectangle(0, 0, 25, 25);
look.put("large", r3);
```

上述代码将 3 个 `Rectangle` 对象添加到该字典中，并使用字符串作为键值。要获取元素，可使用方法 `get`，并指定相应的键值：

```
Rectangle r = (Rectangle)look.get("medium");
```

您还可以使用方法 `remove()` 删除指定键值对应的元素：

```
look.remove("large");
```

您可以使用方法 `size()` 来获悉结构中有多少元素，就像类 `Vector` 一样：

```
int size = look.size();
```

也可以使用方法 `isEmpty()` 来检查结构是否为空:

```
boolean isEmpty = look.isEmpty();
```

### 8.2.7 散列表

类 `Hashtable` 是从 `Dictionary` 派生而来的, 它实现了接口 `Map`, 并提供了键映射数据结构的完整实现。散列表让您能够基于某种类型的键值来存储数据, 并具有由负载系数定义的效率。负载系数是一个  $0.0 \sim 1.0$  的浮点数, 它决定了散列表如何以及何时为更多的元素分配空间。

与矢量一样, 散列表也有容量 (分配的内存量)。散列表通过将表的当前大小同容量和负载系数的乘积进行比较来分配内存。如果散列表的大小超过了这个乘积, 散列表将通过重新散列 (`rehash`) 来增加容量。

负载系数越接近 1.0, 内存使用效率越高, 但代价是查找元素的时间越长。同样, 负载系数越接近于 0.0, 查找的效率越高, 但浪费的内存越多。决定散列表的负载系数时, 取决于您将如何使用散列表以及您看重的是性能还是内存的使用效率。

可以用 3 种方式之一来创建散列表。第一个构造函数创建默认的散列表:

```
Hashtable hash = new Hashtable();
```

第二个构造函数创建具有指定初始容量的散列表:

```
Hashtable hash = new Hashtable(20);
```

最后, 第三个构造函数创建具有指定初始容量和负载系数的散列表:

```
Hashtable hash = new Hashtable(20, 0.75F);
```

在 `Map` 中定义的所有抽象方法都在类 `Hashtable` 中实现了。此外, 类 `Hashtable` 实现了一些其他的方法, 用于实现散列表特有的功能。其中之一是方法 `clear()`, 它删除散列表中的所有键值和元素:

```
hash.clear();
```

方法 `contains()` 检查散列表中是否包含指定的对象。该方法在散列表中查找对象而不是键值。下述代码演示了如何使用方法 `contains()`:

```
boolean isThere = hash.contains(new Rectangle(0, 0, 5, 5));
```

与 `contains()` 类似, 方法 `containsKey()` 也用于查找散列表, 但它基于键值而不是对象。

```
boolean isThere = hash.containsKey("Small");
```

正如前面指出的, 当散列表认为必须增加其容量时, 将重新进行散列。您可以调用方法 `rehash()` 来强行重新散列:

```
hash.rehash();
```

散列表的实际用处在于, 它能够表示那些根据值进行查找或引用时太耗时的数据。换句话说, 处理复杂数据时, 如果使用键值比对数据对象本身进行比较来访问这些数据的效率更高, 则散列表将很有用。

另外, 散列表通常计算元素的键值, 这被称为散列码。例如, 字符串可以有一个整数散列码, 它能够唯一地表示该字符串。当一组字符串被存储在散列表中后, 该表可以通过整数散列码 (而不是字符串本身的内容) 来访问这些字符串。这样, 搜索和检索时, 效率将高得多。

散列码 (hash code) 是通过计算得到的键值, 它唯一地标识了散列表中的每个元素。

在 Java 类库中, 大量地使用了这种计算和使用散列码以存储和引用对象的技术。所有类的超类 `Object` 定义了一个 `hashCode()` 方法, 在大多数标准的 Java 类中都覆盖了这个方法。定义了方法 `hashCode()` 的类都能够通过散列表进行高效地存储和访问。要被散列, 类还必须实现方法 `equals()`, 该方法定义了一种指出两个对象是否相等的方式。`euqals()` 方法通常对类中定义的所有成员变量进行直接比较。

接下来的一个工程将使用散列表来实现一个购物应用程序。

应用程序 ComicBooks 将根据连环画图书的基价 (base value) 和新旧程度进行定价。新旧程度被描述为: 崭新 (mint)、九成新 (near mint)、很新、新、一般、很旧。

每种情况对价格的影响如下:

- 崭新 (Mint): 价格为基价的 3 倍。
- 九成新 (Near mint): 价格为基价的 2 倍。
- 很新 (Very fine): 价格为基价的 1.5 倍。
- 新 (Fine): 价格为基价。
- 一般 (Good): 价格为基价的 0.5 倍。
- 很旧 (Poor): 价格为基价的 0.25 倍。

为将诸如 “mint” 和 “very fine” 等文本同数字值关联起来, 必须将其存储到散列表中。散列表中的键值为有关新旧程度的描述, 而值为浮点数, 如 3.0、1.5 和 0.25 等。

请在 Java 编辑器中输入程序清单 8.3 中的代码, 然后将其保存为 ComicBooks.java。

程序清单 8.3 完整的 ComicBooks.java 源代码

```

1: import java.util.*;
2:
3: public class ComicBooks {
4:
5:     public ComicBooks() {
6:     }
7:
8:     public static void main(String[] arguments) {
9:         // set up hash table
10:        Hashtable quality = new Hashtable();
11:        float price1 = 3.00F;
12:        quality.put("mint", price1);
13:        float price2 = 2.00F;
14:        quality.put("near mint", price2);
15:        float price3 = 1.50F;
16:        quality.put("very fine", price3);
17:        float price4 = 1.00F;
18:        quality.put("fine", price4);
19:        float price5 = 0.50F;
20:        quality.put("good", price5);
21:        float price6 = 0.25F;
22:        quality.put("poor", price6);
23:        // set up collection
24:        Comic[] comix = new Comic[3];
25:        comix[0] = new Comic("Amazing Spider-Man", "1A", "very fine",
26:            9240.00F);
27:        comix[0].setPrice( (Float) quality.get(comix[0].condition) );
28:        comix[1] = new Comic("Incredible Hulk", "181", "near mint",
29:            1325.00F);
30:        comix[1].setPrice( (Float) quality.get(comix[1].condition) );
31:        comix[2] = new Comic("Cerebus", "1A", "good", 45.00F);
32:        comix[2].setPrice( (Float) quality.get(comix[2].condition) );
33:        for (int i = 0; i < comix.length; i++) {
34:            System.out.println("Title: " + comix[i].title);
35:            System.out.println("Issue: " + comix[i].issueNumber);
36:            System.out.println("Condition: " + comix[i].condition);
37:            System.out.println("Price: $" + comix[i].price + "\n");
38:        }
39:    }
40: }
41:
42: class Comic {
43:     String title;
44:     String issueNumber;
45:     String condition;
46:     float basePrice;
47:     float price;
48:
49:     Comic(String inTitle, String inIssueNumber, String inCondition,

```

```

50:         float inBasePrice) {
51:
52:             title = inTitle;
53:             issueNumber = inIssueNumber;
54:             condition = inCondition;
55:             basePrice = inBasePrice;
56:         }
57:
58:         void setPrice(float factor) {
59:             price = basePrice * factor;
60:         }
61: }

```

编译应用程序 ComicBooks 时也会出现前面介绍的警告“未经检查或不安全的操作”，下一节将介绍如何解决这种问题。

运行该应用程序时，其输出将如下：

```

Title: Amazing Spider-Man
Issue: 1A
Condition: very fine
Price: $13860.0

```

```

Title: Incredible Hulk
Issue: 181
Condition: near mint
Price: $2650.0

```

```

Title: Cerebus
Issue: 1A
Condition: good
Price: $22.5

```

该应用程序被实现为两个类：一个名为 ComicBooks 的应用程序类和一个名为 Comic 的助手类。

在该应用程序中，散列表是在第 9~22 行创建的。

首先，第 10 行创建了散列表。

然后，创建了一个名为 price1 的 Float 对象，其值为 3.00。这个值被加入到散列表中，并将其与键值“mint”关联起来。和其他数据结构一样，散列表也只能存储对象——浮点数将通过自动封装自动转换为 Float 对象。

对于其他每种图书（从九成新到很旧），重复执行上述过程。

设置好散列表后，创建了一个名为 comix 的 Comic 对象，用于存储要销售的每本连环画图书。

调用 Comic 构造函数时，提供了 4 个参数：书名、书号、新旧程度和基价。其中前 3 个参数是字符串，最后一个浮点数。

创建 Comic 对象后，调用其 setPrice(Float)方法来根据图书的新旧程度设置价格，如下例所示（第 27 行）：

```
comix[0].setPrice( (Float)quality.get(comix[0].condition) );
```

然后，调用散列表的 get(String)方法，并将图书的新旧程度（一个用作散列表键值的 String）作为参数传递给它。该方法返回一个 Object，它表示与指定的键值相关联的值。在第 27 行，由于 comix[0].condition 等于“mint”，因此 get() 返回一个包含浮点数 3.00F 的 Object。

由于 get() 返回一个 Object，因此必须将其强制转换为 Float。

对于其他两种书，重复上述过程。

第 33~38 行将 comix 数组中存储的关于每本连环画的信息显示出来。

Comic 类是在第 42~61 行定义的。其中有 5 个实例变量：String 对象 title、issueNumber 和 condition 以及浮点变量 baseprice 和 price。

这个类的构造函数方法位于第 49~56 行，它将 4 个变量的值设置为传递给它的参数。

第 58~60 行的 setPrice(Float)方法设置连环画图书的价格。传递给该方法的参数是一个 Float 对象，它被转换为相应的 float 值。接下来的一行计算连环画图书的价格，这是通过将该 float 的值同基价相乘

得到的。因此，如果图书的基价为 1 000 美元，乘数为 2.0，则其定价将为 2 000 美元。

散列表是一种功能强大的、操纵大量数据的数据结构。在 Java 类库中，通过对象 Object 广泛地支持了散列表，这说明它对于 Java 编程而言是至关重要的。

### 8.3 泛型

本章介绍的数据结构无疑是 Java 类库中最基本的实用类。

无论您需要开发的是哪类程序，java.util 包中的散列表、矢量、堆栈和其他结构都很有用。几乎每个软件程序都需要以某种方式对数据进行处理。

这些数据结构也非常适合用于编写适用于各种对象类的代码。为操纵矢量而编写的方法，也可用于对字符串、字符串缓冲区、字符数组和其他表示文本的对象执行相同的功能。会计程序中的方法可以接受表示整数、浮点数和其他数学类的对象，并使用它们来计算结余。

这种灵活性是要付出代价的：如果数据结构能够处理任何类型的对象，则当程序员错误地使用这种数据结构时，Java 编译器将不会提出警告。

例如，应用程序 ComicBook 使用一个名为 quality 的散列表将新旧程度描述（如崭新和新）同价格乘数关联起来。下面是针对九成新的语句：

```
quality.put("near mint", 1.50F);
```

根据设计，散列表 quality 应该只能（以 Float 对象的方式）存储浮点数。然而，不管在这个类中将什么样的值加入到散列表中，这个类都将通过编译。程序员可能无意间将字符串加入到散列表中，如下面的语句：

```
quality.put("near mint", "1.50");
```

这个类仍将通过编译，但在运行时，执行到下述语句时将发生 ClassCastException 错误，进而停止运行：

```
comix[1].setPrice( (Float) quality.get(comix[1].condition) );
```

发生这种错误的原因在于，上述语句试图将散列表中的“1.50”转换为一个 Float 对象。

鉴于显而易见的原因，对程序员而来，运行阶段错误要比编译器错误棘手得多。编译器错误让您无法继续跟踪，必须修改错误后才能继续；而运行阶段错误可能进入代码，而程序员对此一无所知，从而给软件用户带来麻烦。

可使用 Java 语言支持的泛型来指定数据结构中期望的类。

期望类信息被加入到将结构赋给变量或使用构造函数来创建结构的语句中。将期望的类用字符<和>括起，并将其放在数据结构名的后面，如下面的语句所示：

```
Vector<Integer> zipCodes = new Vector<Integer>;
```

上述语句创建一个用于存储 Integer 对象的 Vector。以这种方式声明矢量后，下述语句将导致编译器错误：

```
zipCodes.add("90210");
zipCodes.add("02134");
zipCodes.add("20500");
```

编译器知道，不能将 String 对象加入到这个矢量中。将元素加入到这个矢量中的正确方式是使用整数值：

```
zipCodes.add(90210);
zipCodes.add(02134);
zipCodes.add(20500);
```

对于支持多种类的数据结构（如散列表），可将这些类的名称用<和>括起，并用逗号将它们隔开。

对于应用程序 ComicBook 中的问题，可通过将第 10 行修改成下面这样来解决：

```
Hashtable<String, Float> quality = new Hashtable<String, Float>();
```

上述语句创建一个分别用 String 对象和 Float 对象作为键值和值的散列表。这样，便不能将字符串

作为值加入到散列表中。编译器错误将指出这种问题。

另外，泛型还使得检索数据结构中的对象更简单——不需要将它们强制转换为所需的类。例如，散列表 `quality` 不再需要使用下面这样的语句将对象强制转换为 `Float` 对象：

```
comix[1].setPrice(quality.get(comix[1].condition));
```

从风格的角度看，在变量声明和构造函数方法中加上泛型好像限制了自由。但习惯使用泛型、自动封装、拆封和新的 `for` 循环后，将发现数据结构使用起来更容易，且不容易出错。

程序清单 8.4 中的 `CodeKeeper2` 使用了泛型和新的 `for` 循环，后者用于遍历诸如矢量等数据结构。

程序清单 8.4 `CodeKeeper2` 的完整源代码

```
1: import java.util.*;
2:
3: public class CodeKeeper2 {
4:     Vector<String> list;
5:     String[] codes = { "alpha", "lambda", "gamma", "delta", "zeta" };
6:
7:     public CodeKeeper2(String[] userCodes) {
8:         list = new Vector<String>();
9:         // load built-in codes
10:        for (int i = 0; i < codes.length; i++) {
11:            addCode(codes[i]);
12:        }
13:        // load user codes
14:        for (int j = 0; j < userCodes.length; j++) {
15:            addCode(userCodes[j]);
16:        }
17:        // display all codes
18:        for (String code : list) {
19:            System.out.println(code);
20:        }
21:    }
22:
23:    private void addCode(String code) {
24:        if (!list.contains(code)) {
25:            list.add(code);
26:        }
27:    }
28:
29:    public static void main(String[] arguments) {
30:        CodeKeeper2 keeper = new CodeKeeper2(arguments);
31:    }
32: }
```

唯一被修改的是第 4 行（使用泛型将矢量声明为用于存储字符串）和第 18~19 行（使用了更简单的 `for` 循环）。

#### 注意

使用泛型可限制数据结构只能存储指定的类，甚至您自己设计的类。例如，在支票簿应用程序中，可定义一个表示个人支票的 `Check` 类，再定义一个只能存储 `Check` 对象的矢量。

## 8.4 总结

本章介绍了多种可用于 Java 程序中的数据结构。

- 位组：一组开/关（布尔）值。
- 堆栈：一种后进先出的数据结构。
- 矢量：可动态地改变大小，并根据需要进行缩放的数组。
- 散列表：使用独一无二的键值存储和检索的对象。



这些数据结构位于 `java.util` 包中，后者是一组对处理数据、日期、字符串和其他信息很有帮助的种类。新增的泛型和用于遍历的 `for` 循环改善了它们的功能。

了解在 Java 中可以使用何种方式来组织数据对于软件开发的方方面面都有帮助。无论您学习这种语言的目的是编写 `servlet`、控制台程序、带图形用户界面的消费者（`consumer`）软件还是其他东西，都将需要使用各种不同的方式来表示数据。

## 8.5 问与答

问：本章的 `HolidaySked` 工程也可以使用布尔数组来实现。这两种方式之间孰优孰劣呢？

答：这取决于具体情况。使用数据结构时，您将发现，实现同一种目标的方式有很多。如果关心程序的大小，则位组优于布尔数组，因为前者占用的内存更少；如果关心的是程序的速度，则诸如布尔型等基本数据类型数组更佳，因为数组的速度要快些。就 `Holiday` 类而言，由于它很小，这种差别几乎可忽略不计，但当您开发健壮的应用程序时，这种决策将带来绝然不同的结果。

问：Java 编译器针对没有使用泛型的数据结构的警告是不合适的：发布一个未经检查或使用不安全操作的类并非好主意。是否还有其他的原因导致根本不在数据结构中使用泛型？

答：在作者看来，编译器有关安全性的新警告有些言过其实。多年来，Java 程序员一直在其类中使用矢量、散列表和其他数据结构来创建能够可靠和安全地运行的软件。不使用泛型意味着需要做更多的工作，来确保程序不会因为错误的类被加入到数据结构中而出现运行阶段错误。

更准确地说，通过使用泛型，数据结构更安全了；而不是说以前的 Java 版本在这方面是不安全的。

作者个人遵循的通用规则是，编写新代码或重新组织和修改旧代码时，将使用泛型；而对于旧代码则不去管它。

## 8.6 小测验

请回答下述问题，以复习本章介绍的内容。

### 8.6.1 问题

1. 下述哪种数据可被存储在散列表中？
  - a. `String`;
  - b. `int`;
  - c. a 和 b 都可以。
2. 创建一个矢量，并将 3 个字符串（`Tinker`、`Evers` 和 `Chance`）加入其中后，调用了方法 `removeElement("Evers")`。下述哪个 `Vector` 方法调用将返回字符串 `"Chance"`？
  - a. `get(1)`;
  - b. `get(2)`;
  - c. `get("Chance")`。
3. 下面的哪个类实现了 `Map` 接口？
  - a. `Stack`;
  - b. `Hashtable`;
  - c. `Bitset`。

**答案**

1. c, 在以前的 Java 版本中, 要将诸如 int 等基本数据类型存储到散列表中, 必须使用对象 (例如, 对于 int, 使用 Integer 对象) 来表示它们的值。但在 Java 6 中, 情况并非如此: 基本数据类型将通过自动封装功能自动转换为相应的对象类。

2. a, 添加或删除数据项后, 矢量中每个数据项的索引值将发生变化。由于 “Evers” 被删除后, “Chance” 将成为 Vector 中的第二个数据项, 因此要获取它, 应调用 get(1)。

3. b。

**8.6.2 认证练习**

下面的问题是 Java 认证考试中可能出现的问题, 请回答该问题, 而不要查看本章的内容, 也不要使用 Java 编译器对代码进行测试。

对于下述代码:

```
public class Recursion {
    public int dex = -1;

    public Recursion() {
        dex = getValue(17);
    }

    public int getValue(int dexValue) {
        if (dexValue > 100)
            return dexValue;
        else
            return getValue(dexValue * 2);
    }

    public static void main(String[] arguments) {
        Recursion r = new Recursion();
        System.out.println(r.dex);
    }
}
```

其输出为:

- a. -1;
- b. 17;
- c. 34;
- d. 136。

**答案 d**

**8.7 练习**

为巩固本章介绍的知识, 请尝试完成下面的练习。

1. 在 ComicBooks 应用程序中新增两种新旧程度: “pristine mint” (这种书的价格应为基价的 5 倍) 和 “coverless” (这种书的价格应为基价的 1/3)。

2. 创建一个应用程序, 使用矢量来实现购物车, 用于存储 Fruit 对象。每个 Fruit 对象都有名称、数量和价格。

## 第 9 章

# 使用 Swing

Java 类库中包含一组名为 Swing 的包，让您能够在 Java 程序中实现图形用户界面，并接收来自键盘、鼠标和其他输入设备的用户输入。

Swing 是 Abstract Windowing Toolkit 的一个扩展，后者在最初的 Java 版本中提供了有限的图形编程支持。

本章您将使用 Swing 来创建包含如下图形用户界面组件的应用程序。

- 框架：可以包含标题栏、菜单栏以及最大化、最小化和关闭按钮的窗口；
- 容器：包含其他组件的界面元素。
- 按钮：可单击的区域，其中包含指出其用途的文本或图形。
- 标签：提供信息的文本或图形。
- 文本框和文本区域：接收键盘输入、编辑文本的窗口。
- 下拉列表：可从下拉式菜单或滚动窗口中选择的一组相关项。
- 复选框和单选按钮：小框或圆圈，可被选中或取消选中。

### 9.1 创建应用程序

Swing 让您能够使用操作系统（如 Windows、Solaris）风格或 Java 特有风格（Ocean 和 Metal）来创建 Java 程序的界面。这些风格被称为外观，因为它们描述了界面的外观及其组件的功能。

Swing 的所有成员都是 javax.swing 包的一部分，后者是 Java 类库中的标准组成部分。要使用 Swing 类，必须用 import 语句导入该类或采用下面的方式导入整个包：

```
import javax.swing.*;
```

用于图形用户界面编程的其他两个包分别是 java.awt (Abstract Windowing Toolkit) 和 java.awt.event (处理用户输入的事件处理类)。

使用 Swing 组件时，实际操纵的是该组件类的对象。您通过调用构造方法来创建组件，然后调用相应的方法来正确地设置组件。

所有 Swing 组件都是抽象类 JComponent 的子类，后者包含用于设置组件大小、修改背景颜色、定义文本字体以及设置工具提示（用户将鼠标指向组件时显示的说明性文字）的方法。

#### 警告

Swing 类与 AWT 的许多超类相同，因此可以在同一个界面中同时使用 Swing 和 AWT 组件。然而，在某些情况下，这两种组件不能在容器中被正确显示。为避免这种问题，最好只使用 Swing 组件，除非编写的是仅限于 Java 1.0 或 Java 1.1 功能的小应用程序——对于每个 AWT 组件，都有相应的 Swing 版本。

组件必须被添加到容器 (container) 中后, 才能被显示在用户界面上。容器是可以放置其他组件的组件。Swing 容器通常可放置在其他容器内, 它是 `java.awt.Container` 的子类。这个类包含用于在容器中添加或删除组件的方法, 并使用布局管理器来排列组件和设置容器边界的内部空白区域。

### 9.1.1 创建界面

要创建 Swing 应用程序, 首先要创建一个表示图形用户界面的类。这个类的对象将被用作容器, 用于放置要显示的其他所有组件。

在许多工程中, 主界面对象要么是一个简单的窗口 (`JWindow` 类), 要么是一个叫做框架的窗口 (`JFrame` 类)。

窗口是一个容器, 可被显示到用户桌面上。简单窗口没有标题栏、最大化、最小化和关闭按钮, 也没有图形用户界面操作系统的窗口中最常见的其他元素。包含这些窗口管理特性的窗口被称为框架 (frame)。

在诸如 Windows 或 MacOS 等图形环境中, 用户能够对它们运行的程序的窗口进行移动、改变大小以及关闭等操作。简单窗口主要在加载程序时显示, 有时出现一个“标题屏幕”, 其中包含程序名、徽标和其他信息。

创建图形 Swing 应用程序的方式之一是, 把界面声明为 `JFrame` 的子类, 如下面的类声明所示:

```
public class FeedReader extends JFrame {
    // ...
}
```

这个类的构造方法需完成下列几项工作:

- 调用超类的构造方法来处理其设置;
- 设置框架窗口的大小, 指定宽度和高度 (单位为像素) 或让 Swing 去选择合适的大小;
- 决定用户关闭窗口时如何办;
- 显示框架。

`JFrame` 类有两个构造函数: `JFrame()` 和 `JFrame(Swing)`, 前者将框架的标题栏设置为空, 后者将其设置为指定的文本。也可以通过调用框架的 `setTitle(Swing)` 方法来设置标题。

框架的大小可以通过调用 `setSize(int, int)`, 并将宽度和高度作为参数来设置。框架大小的单位为像素, 因此如果您调用 `setSize(650, 550)`, 则被显示时该框架将占据分辨率为  $800 \times 600$  的大部分屏幕。

#### 注意

也可以调用方法 `setSize(Dimension)` 来设置框架的大小。Dimension 是 `java.awt` 包中的一个类, 表示用户界面组件的宽度和高度。调用构造函数 `Dimension (int, int)` 来创建一个 Dimension 对象, 它表示参数指定的宽度和高度。

另一种设置框架大小的方式是, 将框架要包含的组件加入其中, 然后调用框架的 `pack()` 方法。这将根据框架包含的组件的大小相应地调整框架的大小。如果框架过大, `pack()` 将其缩小到刚好能够显示其中的组件; 如果框架过小 (或大小根本没有被设置), 则 `pack()` 将其扩大到刚好能够显示其中的组件。

被创建时, 框架是不可见的。可以调用方法 `show()` (不提供任何参数) 或方法 `setVisible(boolean)` (将字面量 `true` 作为参数) 来使之可见。

如果希望框架被创建后就被显示出来, 可在构造方法中调用上述两个方法之一。也可以让框架不可见, 由使用该框架的类通过调用 `show()` 或 `setVisible(true)` 来显示它。要隐藏框架, 可调用 `setVisible(false)`。

默认情况下, 框架被显示后将被放在计算机桌面的左上角。可以通过调用方法 `setBounds(int, int, int, int)` 来指定其他位置, 该方法的前两个参数是框架的左上角 (x, y) 坐标, 后两个参数是框架的宽度和高度。

下面的类表示了一个  $300 \times 100$  的框架, 其标题栏为 “Edit Payroll”:

```

public class Payroll extends javax.swing.JFrame {
    public Payroll() {
        super("Edit Payroll");
        setSize(300, 100);
        setVisible(true);
    }
}

```

每个框架的标题栏上都有最大化、最小化和关闭按钮，您的系统中运行的其他软件界面上也有这样的控件。

当框架被关闭时，正常行为是让应用程序继续执行，如果框架是应用程序的主图形用户界面，这将导致用户无法终止程序。

要修改这种行为，必须调用框架的方法 `setDefaultCloseOperation()`，并将下面 4 个 `JFrame` 类变量之一作为参数。

- `EXIT_ON_CLOSE`：框架被关闭时退出程序。
- `DISPOSE_ON_CLOSE`：框架被关闭时，释放框架对象，并继续运行应用程序。
- `DO_NOTHING_ON_CLOSE`：打开框架窗口并继续运行程序。
- `HIDE_ON_CLOSE`：关闭框架窗口，并继续运行程序。

为防止用户关闭框架窗口，可在其构造方法中添加如下语句：

```
setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
```

如果创建的框架将用作应用程序的主用户界面，则其行为可能是 `EXIT_ON_CLOSE`：关闭框架，然后结束应用程序。

### 9.1.2 开发框架

程序清单 9.1 是一个简单的应用程序，它显示一个  $300 \times 100$  像素的框架窗口。这个类可以用作任何图形用户界面应用程序的框架（framework）。

**程序清单 9.1 完整的 SimpleFrame.java 源代码**

```

1: import javax.swing.JFrame;
2:
3: public class SimpleFrame extends JFrame {
4:     public SimpleFrame() {
5:         super("Frame Title");
6:         setSize(300, 100);
7:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8:         setVisible(true);
9:     }
10:
11:     public static void main(String[] arguments) {
12:         SimpleFrame sf = new SimpleFrame();
13:     }
14: }

```

编译并运行该应用程序时，将看到如图 9.1 所示的框架。

应用程序 `SimpleFrame` 没有太多需要说明的地方：除了标题栏上的最大化、最小化和关闭按钮（见图 9.1）外，这个图形用户界面没有包含其他组件。本章后面将加入其他的组件。



图 9.1 显示框架

在方法 `main()` 中, 第 11~13 行创建了一个 `SimpleFrame` 对象。如果没有在构造函数中将框架显示出来, 可以在方法 `main()` 中调用 `sf.setVisible(true)` 来显示该框架。

创建框架的用户界面所涉及的工作是在构造方法 `SimpleFrame()` 中完成的。如果要向框架中添加组件, 可在这个构造函数中创建它们, 并将它们加入到框架中。

使用 `JWindow` 创建窗口与使用 `Swing` 的框架类似, 但不能提供标题或关闭窗口。

程序清单 9.2 中的应用程序创建并打开一个窗口, 将其大小从  $10 \times 10$  修改为  $400 \times 400$ 。

程序清单 9.2 完整的 `SimpleWindow.java` 源代码

```
1: import javax.swing.JWindow;
2:
3: public class SimpleWindow extends JWindow {
4:     public SimpleWindow() {
5:         super();
6:         setBounds(400, 300, 10, 10);
7:         setVisible(true);
8:     }
9:
10:    public static void main(String[] arguments) {
11:        SimpleWindow sw = new SimpleWindow();
12:        for (int i = 10; i < 400; i++) {
13:            sw.setBounds(400 - (i/2), 300 - (i/2), i, i);
14:        }
15:    }
16: }
```

在这个应用程序中, 第 6 行的 `setBounds(400, 300, 10, 10)` 将窗口的大小设置为  $10 \times 10$ , 并将其左上角显示在  $(400, 300)$  处。

第 12~14 行的 `for` 循环在每次迭代时都修改窗口的大小, 并移动其位置。随着循环的进行, 窗口的大小将从  $10 \times 10$  变为  $400 \times 400$ 。您可按 `Ctrl + C` 关闭窗口和应用程序。

### 9.1.3 显示启动画面

`JWindow` 组件没有装饰, 这使其适合作启动画面 (splash page) —— 应用程序加载时显示的图形或文本。

Java 6 新增了一种更好、更快地实现这种方法, 这是通过将图形指定为应用程序的启动画面实现的。这种图形甚至在 Java 解释器加载前加载, 并在应用程序开始运行前消失。

从命令行运行应用程序时, 可使用属性 `-splash` 指定启动画面。下面演示了如何给 `SimpleWindow` 指定启动画面:

```
java -splash:lighthouse.jpg SimpleWindow
```

#### 注意

“注意” 如果需要使用图形来测试这种功能, 可从人民邮电出版社网站 [www.ptpress.com.cn](http://www.ptpress.com.cn) 下载本书资源。其中的 `lighthouse.jpg` 是 Steve Cadman 拍摄的一个挪威灯塔的照片。

### 9.1.4 创建组件

创建图形用户界面是一种非常不错的获得 Java 对象使用经验的途径, 因为界面中的每个组件都有对应的类。

要使用界面组件, 可以创建该组件类的一个对象, 您已经使用过容器类 `JFrame` 和 `JWindow`。

最容易使用的组件之一是 `JButton`, 它表示可单击的按钮。

在大多数程序中, 按钮都将触发某种操作: 单击“安装”将开始安装软件; 单击“笑脸”按钮将

重新开始“扫雷”游戏；单击最小化按钮可防止老板发现您正玩“扫雷”游戏等。

Swing 按钮可以有文本标签、图形图标或兼而有之。

按钮类的构造方法包括：

- JButton(String)：创建一个带指定的文本的按钮。
- JButton(Icon)：创建一个包含指定图标的按钮。
- JButton(String, Icon)：创建一个包含指定文本和图标的按钮。

下面的语句创建了 3 个包含文本标签的按钮：

```
JButton play = new JButton("Play");
JButton stop = new JButton("Stop");
JButton rewind = new JButton("Rewind");
```

图形按钮将在本章后面介绍。

### 9.1.5 将组件加入到容器中

显示用户界面组件（如 Java 程序中的按钮）之前，必须将它加入到容器中，然后显示容器。

要将组件加入到简单容器中，可以调用容器的 add(Component)方法，并将该组件作为参数（Swing 中的所有用户界面组件都是从 java.awt.Componet 派生而来的）。

最简单的 Swing 容器是面板（JPanel 类）。下面的代码创建了一个按钮并将它加入到面板中：

```
JButton quit = new JButton("Quit");
JPanel panel = new JPanel();
panel.add(quit);
```

将组件加入到框架和窗口中的方法与此相同。

#### 注意

在以前的 Java 版本中，组件不能直接被加入到框架中，而是被放置在容器的内容窗格中。虽然这种技术不再是必须的，但您很可能在代码中看到这种情况。您可以使用面板来表示框架的内容窗格，并使用面板的 add(Component)方法将组件加入到面板中。然后，调用框架的 setContentPane(Container)方法，并将面板作为参数传递给它，这样面板将成为框架的内容窗格。对于窗口也可以这样做。

程序清单 9.3 中的程序使用了本章前面创建的应用程序框架。它创建了一个面板，然后将 3 个按钮加入到面板中，并将面板加入到框架中。

程序清单 9.3 完整的 ButtonFrame.java 源代码

```
1: import javax.swing.*;
2:
3: public class ButtonFrame extends JFrame {
4:     JButton load = new JButton("Load");
5:     JButton save = new JButton("Save");
6:     JButton unsubscribe = new JButton("Unsubscribe");
7:
8:     public ButtonFrame() {
9:         super("Button Frame");
10:        setSize(140, 170);
11:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:        JPanel pane = new JPanel();
13:        pane.add(load);
14:        pane.add(save);
15:        pane.add(unsubscribe);
16:        add(pane);
17:        setVisible(true);
18:    }
19:
20:    public static void main(String[] arguments) {
21:        ButtonFrame bf = new ButtonFrame();
22:    }
23: }
```



运行该应用程序时，将显示一个包含 3 个按钮的框架，如图 9.2 所示。

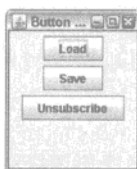


图 9.2 应用程序 ButtonFrame

ButtonFrame 类有 3 个实例变量：JButton 对象 load、save 和 unsubscribe。

第 12~15 行创建了一个新的 JPanel 对象，并调用其 add() 方法将 3 个按钮加入到其中。将按钮加入到面板中后，第 16 行调用框架的 add() 方法并将面板作为参数，将其加入到框架中。

#### 注意

如果您单击这些按钮，绝对不会有任何事情发生。第 12 章将介绍如何对用户单击按钮做出响应。

## 9.2 使用组件

除了按钮和前面介绍过的容器外，Swing 还提供了 20 多个不同的用户界面组件。在本章余下的内容和第 10 章中，将介绍其中的很多组件。

所有 Swing 组件都是从超类 javax.swing.JComponent 派生而来的，并从中继承了一些方法，这些方法很有用。

方法 setEnabled(boolean) 在参数为 true 时启用组件，在参数为 false 时禁用组件。默认情况下，组件是被启用的。要接收用户输入，组件必须处于启用状态。许多组件被禁用时，外观将发生变化，以此来表明它们当前不可用。例如，JButton 被禁用时，其边框和文本将呈灰色。要检查组件是否被启用，可调用方法 isEnabled()，该方法返回一个布尔值。

方法 setVisible(boolean) 处理所有组件的方式与处理容器相同。使用 true 来显示组件，使用 false 来隐藏它。另外，也可以使用布尔型方法 isVisible()。

方法 setSize(int, int) 将组件的宽度和高度设置为参数指定的值，setSize(Dimension) 使用一个 Dimension 对象来完成同样的工作。对大多数组件而言，不必设置其大小——默认的设置通常是可行的。要获悉组件的大小，可调用方法 getSize()，它返回了一个 Dimension 对象，该对象的实例变量 height 和 width 指出了组件的大小。

正如您将看到的，相似的 Swing 组件也有其他一些相同的方法，如用于文本组件的 setText() 和 getText()，用于存储数据的组件的 setValue() 和 getValue()。

#### 警告

使用 Swing 组件时，一种常见的错误是将组件加入容器中后再去设置其属性。务必在将组件加入到面板和其他容器之前，设置好组件的所有属性。

### 9.2.1 图标

Swing 支持在按钮和其他提供标签的组件上使用对象 ImageIcon。图标是一个小图片，可被放置在按钮、标签或其他用户界面元素上，以说明其用途，例如垃圾桶和回收站图标用于删除文件、文件夹图标用于存储文件等。



要创建 `ImageIcon` 对象，可调用其构造函数，并将图形文件名作为参数传递给它。下面的例子装载文件 `subscribe.gif` 中的图标，并创建了一个以该图标作为标签的 `JButton`：

```
ImageIcon subscribe = new ImageIcon("subscribe.gif");
JButton button = new JButton(subscribe);
JPanel pane = new JPanel();
pane.add(button);
add(pane);
setVisible(true);
```

程序清单 9.4 中的 Java 应用程序创建了 4 个带图标和文本标签的按钮，然后将这些按钮加入到一个面板中，再将面板加入到一个框架中。

该应用程序和本周的众多其他范例程序是开发大型应用程序的基础。

这个应用程序是一个 RSS 新闻阅读器——每隔 1 小时从网站中检索新闻标题和其他数据，让用户能够快速决定阅读还是跳过。

#### 程序清单 9.4 完整的 `IconFrame.java` 源代码

```
1: import javax.swing.*;
2:
3: public class IconFrame extends JFrame {
4:     JButton load, save, subscribe, unsubscribe;
5:
6:     public IconFrame() {
7:         super("Icon Frame");
8:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9:         JPanel panel = new JPanel();
10:        // create icons
11:        ImageIcon loadIcon = new ImageIcon("load.gif");
12:        ImageIcon saveIcon = new ImageIcon("save.gif");
13:        ImageIcon subscribeIcon = new ImageIcon("subscribe.gif");
14:        ImageIcon unsubscribeIcon = new ImageIcon("unsubscribe.gif");
15:        // create buttons
16:        load = new JButton("Load", loadIcon);
17:        save = new JButton("Save", saveIcon);
18:        subscribe = new JButton("Subscribe", subscribeIcon);
19:        unsubscribe = new JButton("Unsubscribe", unsubscribeIcon);
20:        // add buttons to panel
21:        panel.add(load);
22:        panel.add(save);
23:        panel.add(subscribe);
24:        panel.add(unsubscribe);
25:        // add the panel to a frame
26:        add(panel);
27:        pack();
28:        setVisible(true);
29:    }
30:
31:    public static void main(String[] arguments) {
32:        IconFrame ike = new IconFrame();
33:    }
34: }
```

该程序的运行情况如图 9.3 所示。



图 9.3 包含带图标的按钮的界面

在第 11~14 行引用的图形可以在本书的下载资源中找到。

应用程序 `IconFrame` 没有设置框架的大小，而是在第 27 行调用方法 `pack()` 来扩展框架，使之刚好能够容纳 4 个按钮。

如果框架被设置为高度大于宽度——例如，在构造函数中调用 `setSize(100, 400)`，按钮将垂直排列。

#### 注意

有些项目中的图标来自 Sun 的 Look and Feel 图形库——一个在您自己的程序中使用的图标集。

### 9.2.2 标签

标签是一个包含说明性文本、图标或两者都有的用户组件。可以使用类 `JLabel` 来创建标签，它通常来说明界面上其他组件的用途，用户不能直接编辑。

要创建标签，可使用下述构造函数。

- `JLabel(String)`: 带指定文本的标签。
- `JLabel(String, int)`: 带指定的文本和对齐方式的标签。
- `JLabel(String, Icon, int)`: 带指定文本、图标和对齐方式的标签。

标签的对齐方式决定了其文本和图标同窗口占据区域之间的对齐关系。`SwingConstants` 接口的 3 个静态类变量被用来指定对齐方式: `LEFT`、`CENTER` 和 `RIGHT`。

标签的内容可以使用方法 `setText(String)` 或 `setIcon(Icon)` 来设置，还可以使用方法 `getText()` 和 `getIcon()` 来获取这些内容。

下面的语句创建了 3 个标签，它们的对齐方式分别是左对齐、居中和右对齐：

```
JLabel feedsLabel = new JLabel("Feeds", SwingConstants.LEFT);
JLabel urlLabel = new JLabel("URL: ", SwingConstants.CENTER);
JLabel dateLabel = new JLabel("Date: ", SwingConstants.RIGHT);
```

### 9.2.3 文本框

文本框是界面上的一片区域，用户可以通过键盘输入、修改其中的内容。文本框是用类 `JTextField` 来表示的，能够处理一行输入。后面将介绍一个类似的组件：文本区域，它能够处理多行输入。

文本框的构造方法有 3 个。

- `JTextField()`: 空文本框。
- `JTextField(int)`: 指定宽度的文本框。
- `JTextField(String, int)`: 指定宽度、包含指定字符串的文本框。

仅当组织界面时不修改组件的大小，文本框的宽度属性才适用。第 11 章介绍布局管理器时，您将对此有更深入的了解。

下面的语句创建两个能够容纳大约 60 个字符的文本框，但前一个文本框为空，而后一个的内容为 `Enter RSS feed URL here`。

```
JTextField rssUrl1 = new JTextField(60);
JTextField rssUrl2 = new JTextField(
    "Enter RSS feed URL here", 60);
```

文本框和文本区域都是从超类 `JTextComponent` 派生而来的，因此有很多相同的方法。

方法 `setEditable(boolean)` 指定文本组件可被编辑（参数为 `true` 时）还是不可被编辑（参数为 `false` 时），而方法 `isEditable()` 返回相应的 `boolean` 值。

方法 `setText(string)` 将文本修改为指定的字符串；方法 `getText()` 将当前的文本作为字符串返回；方法 `getSelectedText()` 返回被选中的文本。

密码框（`password field`）也是一个文本框，它将用户输入的字符隐藏起来。密码框通常由类 `JPasswordField` 表示，这是 `JTextField` 的一个子类。`JPasswordField` 的构造方法接受的参数与其父类的

构造方法相同。

创建了密码框后，可调用方法 `setEchoChar(char)` 来使用指定的字符隐藏输入。

下面的语句创建了一个密码框，并将其回显字符设置为#：

```
JPasswordField codePhrase = new JPasswordField(20);
codePhrase.setEchoChar('#');
```

## 9.2.4 文本区域

文本区域 (text areas) 是能够处理多行输入的可编辑文本框，由类 `JTextArea` 实现。

`JTextArea` 包含如下两个构造方法。

- `JTextArea (int, int)`：行数和列数为指定值的文本区域。
- `JTextArea (String, int, int)`：行数和列数为指定值，且包含指定文本的文本区域。

与文本框一样，也可以调用文本区域的方法 `getText()`、`getSelectedText()` 和 `setText(String)`。另外，方法 `append(String)` 将指定的文本加到当前文本的末尾，方法 `insert(String, int)` 将指定的文本插入到指定的位置。

方法 `setLineWrap(boolean)` 指定文本在达到组件边界时是否自动换行，如果参数为 `true`，将自动换行。

方法 `setWrapStyleWord(boolean)` 指定如何换行：是将当前单词（参数为 `true` 时）还是当前字符（参数为 `false` 时）换到下一行。

程序清单 9.5 中的应用程序 `Authenticator` 使用了几个 Swing 组件来收集用户的输入：一个文本框、一个密码框和一个文本区域，同时使用标签来指明每个组件的用途。

程序清单 9.5 完整的 `Authenticator.java` 源代码

```
1: import javax.swing.*;
2:
3: public class Authenticator extends javax.swing.JFrame {
4:     JTextField username = new JTextField(15);
5:     JPasswordField password = new JPasswordField(15);
6:     JTextArea comments = new JTextArea(4, 15);
7:     JButton ok = new JButton("OK");
8:     JButton cancel = new JButton("Cancel");
9:
10:    public Authenticator() {
11:        super("Account Information");
12:        setSize(300, 220);
13:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14:
15:        JPanel pane = new JPanel();
16:        JLabel usernameLabel = new JLabel("Username: ");
17:        JLabel passwordLabel = new JLabel("Password: ");
18:        JLabel commentsLabel = new JLabel("Comments: ");
19:        comments.setLineWrap(true);
20:        comments.setWrapStyleWord(true);
21:        pane.add(usernameLabel);
22:        pane.add(username);
23:        pane.add(passwordLabel);
24:        pane.add(password);
25:        pane.add(commentsLabel);
26:        pane.add(comments);
27:        pane.add(ok);
28:        pane.add(cancel);
29:        add(pane);
30:        setVisible(true);
31:    }
32:
33:    public static void main(String[] arguments) {
34:        Authenticator auth = new Authenticator();
35:    }
36: }
```



图 9.4 显示了该程序的运行情况，使用星号屏蔽了密码。如果调用文本框的 `setEchoChar(char)` 方法时没有指定回显字符，将默认使用星号。

这个应用程序中的文本区域的行为可能与您预期的不同。到达区域底部后，如果用户继续输入文本，文本区域将增大以提供更大的输入空间。下一节将介绍如何添加滚动条，以防文本区域改变大小。

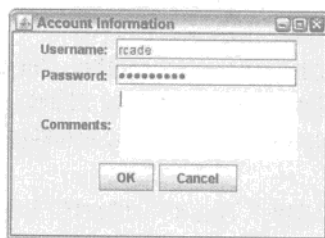


图 9.4 应用程序 Authenticator

### 9.2.5 可滚动窗格

Swing 中的文本区域不包含水平滚动条或垂直滚动条，单独使用这种组件时，无法加入水平滚动条和垂直滚动条。

Swing 通过一种新的容器——`JScrollPane`——来支持滚动条，它可用来放置任何可滚动的组件。在可滚动窗格的构造方法中，可以将其与组件关联起来。可以使用下面的任何一个构造方法。

- `JScrollPane(Component)`：包含指定组件的可滚动窗格。
- `JScrollPane(Component,int,int)`：包含指定的组件，并带垂直滚动条和水平滚动条的可滚动窗格。

滚动条是通过使用接口 `ScrollPaneConstants` 的静态类变量来配置的。对于垂直滚动条，可以使用下述变量之一：

- `VERTICAL_SCROLLBAR_ALWAYS`;
- `VERTICAL_SCROLLBAR_AS_NEEDED`;
- `VERTICAL_SCROLLBAR_NEVER`。

同样，对于水平滚动条，也有 3 个类似的变量。

创建包含组件的可滚动窗格后，应将该窗格（而不是其中的组件）加入到容器中。

下面的例子创建了一个文本区域（它带垂直滚动条，但没有水平滚动条），然后将它加入到内容窗格中：

```
JPanel pane = new JPanel();
JTextArea comments = new JTextArea(4, 15);
JScrollPane scroll = new JScrollPane(comments,
    ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
    ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
pane.add(scroll);
setContentPane(pane);
```

### 9.2.6 复选框和单选按钮

接下来介绍复选框和单选按钮，这两个组件都只有两个可能的取值：被选中或没有被选中。

复选框通常用于进行简单的是/否或开/关选择。单选按钮被组合在一起，这样每次只有其中的一个可被选中。

复选框（`JCheckBox` 类）是带标签或不带标签的框，被选中时，框中将有一个复选标记，否则为空。单选按钮（`JRadioButton` 类）是一个圆圈，被选中时，将有一个圆点，否则为空。

JCheckBox 和 JRadioButton 类从同一个超类那里继承了一些很有用的方法。

- setSelected(boolean): 如果参数为 true, 则选中组件, 否则不选中。
- isSelected(): 返回一个布尔值, 指出组件是否被选中。

类 JCheckBox 包含下述构造函数:

- JCheckBox(String): 带指定文本标签的复选框。
- JCheckBox(String, boolean): 带指定文本标签的复选框, 如果第二个参数为 true, 则被选中。
- JCheckBox(Icon): 带指定图标标签的复选框。
- JCheckBox(Icon, boolean): 带指定图标标签的复选框, 如果第二个参数为 true, 则被选中。
- JCheckBox(String, Icon): 带指定文本标签和图标标签的复选框。
- JCheckBox(String, Icon, boolean): 带指定文本标签和图标标签的复选框, 如果第三个参数为 true, 则被选中。

类 JRadioButton 也包含接受同样参数并具备相同功能的构造函数。

复选框和单选按钮通常是非互斥的, 即如果一个容器中有 5 个复选框, 5 个都可以同时被选中或不被选中。为使单选按钮互斥, 必须将相关的组件分组。

要将多个单选按钮组织成一组, 只允许每次选中其中一个, 可以创建一个 ButtonGroup 对象, 如下面的语句所示:

```
ButtonGroup choice = new ButtonGroup();
```

对象 ButtonGroup 跟踪组中所有单选按钮, 可以调用方法 add(Component) 将指定的组件加入到组中。

下面的例子创建了一个组, 其中包含两个单选按钮:

```
ButtonGroup saveFormat = new ButtonGroup();
JRadioButton s1 = new JRadioButton("OPML", false);
saveFormat.add(s1);
JRadioButton s2 = new JRadioButton("XML", true);
saveFormat.add(s2);
```

对象 saveFormat 用来将单选按钮 r1 和 r2 组织在一起。带标签 XML 的对象 s2 被选中。一次只能有一个成员被选中——如果一个组件被选中, 对象 ButtonGroup 将确保组内的其他组件不被选中。

程序清单 9.6 中的应用程序将 4 个单选按钮组织成一组。

#### 程序清单 9.6 完整的 FormatFrame.java 源代码

```
1: import javax.swing.*;
2:
3: public class FormatFrame extends JFrame {
4:     JRadioButton[] teams = new JRadioButton[4];
5:
6:     public FormatFrame() {
7:         super("Choose an Output Format");
8:         setSize(320, 120);
9:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10:        teams[0] = new JRadioButton("Atom");
11:        teams[1] = new JRadioButton("RSS 0.92");
12:        teams[2] = new JRadioButton("RSS 1.0");
13:        teams[3] = new JRadioButton("RSS 2.0", true);
14:        JPanel panel = new JPanel();
15:        JLabel chooseLabel = new JLabel(
16:            "Choose an output format for syndicated news items.");
17:        panel.add(chooseLabel);
18:        ButtonGroup group = new ButtonGroup();
19:        for (int i = 0; i < teams.length; i++) {
20:            group.add(teams[i]);
21:            panel.add(teams[i]);
22:        }
23:        add(panel);
24:        setVisible(true);
25:    }
26: }
```



```

27:     public static void main(String[] arguments) {
28:         FormatFrame ff = new FormatFrame();
29:     }
30: }

```

图 9.5 显示了该应用程序运行的情况。4 个 JRadioButton 对象被存储在数组中，在第 19~22 行的 for 循环中，首先将每个元素加入到按钮组中，然后将它加入到面板中。循环结束后，该面板被用作应用程序的内容窗格。

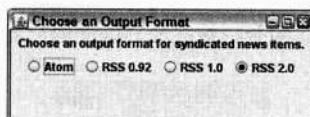


图 9.5 应用程序 FormatFrame

选中其他单选按钮后，原来被选中的单选按钮将不再被选中。

### 9.2.7 组合框

Swing 类 JComboBox 可用于创建组合框，后者提供一个下拉式菜单，用户可以选择一项。当这种组合框未被使用时，菜单被隐藏，这样它在图形用户界面上占用的空间将更小。

创建组合框的步骤如下：

1. 调用构造函数 JComboBox()，且不提供任何参数。
2. 调用组合框的方法 addItem(Object)，将选项加入到列表中。

在组合框中，用户只能选择下拉式菜单中的一项。调用方法 setEditable()，并提供参数 true，可使组合框支持输入文本。组合框因这种特性而得名——它提供了下拉式菜单和文本框。

类 JComboBox 有几个可用于控制下拉列表或组合框的方法。

- getItemAt(int)：返回位于整数参数指定的索引位置的选项文本。与数组一样，选择列表第一项的索引为 0，第二项为 1，依次类推。
- getItemCount()：返回列表中的选项数目。
- getSelectedIndex()：返回当前选项的索引。
- getSelectedItem()：返回当前选项的文本。
- setSelectedIndex(int)：选中索引指定的选项。
- setSelectedIndex(Object)：选中指定的对象。

在程序清单 9.7 中的应用程序 FormatFrame2 修改了前一个范例程序，它使用不可编辑的组合框，让用户能够选择 4 个选项中的一个。

#### 程序清单 9.7 完整的 FormatFrame2.java 源代码

```

1: import javax.swing.*;
2:
3: public class FormatFrame2 extends JFrame {
4:     String[] formats = { "Atom", "RSS 0.92", "RSS 1.0", "RSS 2.0" };
5:     JComboBox formatBox = new JComboBox();
6:
7:     public FormatFrame2() {
8:         super("Choose a Format");
9:         setSize(220, 150);
10:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11:        JPanel pane = new JPanel();
12:        JLabel formatLabel = new JLabel("Output formats:");
13:        pane.add(formatLabel);

```

```

14:         for (int i = 0; i < formats.length; i++)
15:             formatBox.addItem(formats[i]);
16:         pane.add(formatBox);
17:         add(pane);
18:         setVisible(true);
19:     }
20:
21:     public static void main(String[] arguments) {
22:         FormatFrame2 ff = new FormatFrame2();
23:     }
24: }

```

图 9.6 显示了该应用程序运行时，组合框被打开，以使用户能够选择其中的一个值。

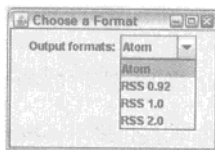


图 9.6 应用程序 FormatFrame2

### 9.2.8 列表

本章介绍的最后一个组件与组合框类似。列表是用类 `JList` 表示的，它让用户能够从中选择一个或多个值。

创建列表时，可以使用数组或矢量的内容来填充。下列是可用的构造函数。

`JList()`: 创建一个空列表。

`JList(Object[])`: 创建一个列表，其内容为指定类（如 `String`）的数组。

`JList(Vector)`: 创建一个列表，其内容为指定的 `Vector` 对象。

对于空列表，可以调用其 `setListData()` 方法来填充，该方法接受一个参数——数组或矢量。

不同于组合框，列表出现在用户界面中时，将显示其内容中的多项，默认显示 8 项。要修改这种设置，可调用方法 `setVisibleRowCount(int)`，并将要显示的项数作为参数。

方法 `getSelectedValues()` 返回一个对象数组，其中包含列表中所有被选中的项。

程序清单 9.8 中的应用程序 `Subscriptions` 显示一个字符串数组中的 8 项。

程序清单 9.8 完整的 `Subscriptions.java` 源代码

```

1: import javax.swing.*;
2:
3: public class Subscriptions extends JFrame {
4:     String[] subs = { "0xDECAFBAD", "Cafe au Lait",
5:         "Hack the Planet", "Ideoplex", "Inessential", "Intertwingly",
6:         "Markpasc", "Postneo", "RC3", "Workbench" };
7:     JList subList = new JList(subs);
8:
9:     public Subscriptions() {
10:         super("Subscriptions");
11:         setSize(150, 300);
12:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13:         JPanel panel = new JPanel();
14:         JLabel subLabel = new JLabel("RSS Subscriptions:");
15:         panel.add(subLabel);
16:         subList.setVisibleRowCount(8);
17:         JScrollPane scroller = new JScrollPane(subList);
18:         panel.add(scroller);
19:         add(panel);
20:         setVisible(true);
21:     }

```

```

22:
23:     public static void main(String[] arguments) {
24:         Subscriptions app = new Subscriptions();
25:     }
26: }

```

图 9.7 是该程序的运行情况。在该应用程序的界面中，有一个显示 8 项的列表，列表的上面是一个标签。第 17~18 行使用了一个滚动窗格，让用户能够滚动列表，以便看到第 9~10 项。

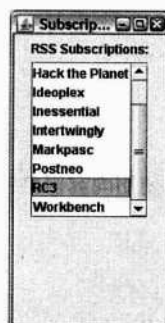


图 9.7 应用程序 Subscriptions

### 9.3 总结

本章首先介绍了 Swing 包，其中的类让您能够在 Java 程序中提供图形用户界面。

您使用了十几个类，创建了诸如按钮、标签和文本框等界面组件。您将这些组件都加入到容器（包括面板、框架、窗口）中。

这种编程工作可能很复杂，Swing 是最大的包，新的 Java 程序员必须使用其中的类。

然而，通过使用诸如文本区域和文本框等组件，您知道 Swing 组件有很多共同的超类，这使得您能够将所学的知识用于其他新的组件、容器，以及接下来的三章中将介绍的其他 Swing 编程领域中。

### 9.4 问与答

问：可以修改按钮或其他组件上的文本的字体吗？

答：类 JComponent 包含一个 setFont(Font) 方法，可用来设置组件上的文本的字体。第 13 章将介绍 Font 对象、颜色和图形。

问：如何知道 Swing 中有哪些组件以及如何使用它们？

答：本书将用两章介绍图形用户界面组件，下一章将更详细地介绍这方面的内容。

问：前一个 Java 版本使用 Metal 外观。我可以继续使用这种风格而不是 Ocean 吗？

答：第 10 章将介绍如何在 Java 类中这样做。另外，还有一个可以指定的系统属性：swing.metalTheme，它导致解释器默认使用 Metal 而不是 Ocean 外观。要采用 Metal 外观，应将该属性的值设置为 steel，如下述命令所示：

```
java -Dswing.metalTheme=steel Authenticator
```

执行上述命令将导致应用程序以 Metal 外观显示。



## 9.5 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 9.5.1 问题

- 下面哪个用户界面组件不是容器？
  - JScrollPane;
  - JTextArea;
  - JWindow。
- 将组件添加到下列哪种容器时不需要使用内容窗格？
  - JPanel;
  - JWindow;
  - JFrame。
- 如果对应用程序的主框架或主窗口使用 `setSize()`，它将出现在桌面的什么地方？
  - 桌面的中央；
  - 上个应用程序出现的地方；
  - 桌面的左上角。

#### 答案

- b，为支持滚动，`JTextArea` 需要一个容器，但它本身并非容器。
- a，`JPanel` 是一种简单容器，没有被划分成窗格，因此可调用其 `add(Component)` 方法将组件直接添加到面板中。不能像框架和窗口那样，单独打开一个面板。
- c，调用 `setSize()` 不会影响窗口在桌面上的位置，要设置框架的位置，必须调用 `setBounds()`，而不是 `setSize()`。

### 9.5.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容或使用 Java 编译器进行试验。

对于下面的代码：

```
import javax.swing.*;

public class Display extends JFrame {
    public Display() {
        super("Display");
        // answer goes here
        JLabel hello = new JLabel("Hello");
        JPanel pane = new JPanel();
        pane.add(hello);
        setContentPane(pane);
        pack();
        setVisible(true);
    }

    public static void main(String[] arguments) {
        Display ds = new Display();
    }
}
```

要使应用程序正常运行，应将 `// answer goes here` 替换为下面哪条语句？



- a. `setSize(300, 200);`
- b. `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
- c. `Display ds = new Display();`
- d. 不需要替换任何语句。

答案 b

## 9.6 练习

为巩固本章介绍的知识，请尝试完成下面的练习：

1. 创建一个带框架的应用程序，其中包含几个 VCR 控件：播放、停止/弹出、倒带、快进和暂停。设置窗口的大小，使得所有组件都显示在同一行中。
2. 创建一个框架，其中包含让用户输入用户名和密码的文本框。



## 第 10 章

# 创建 Swing 界面

虽然计算机可以在命令行环境（如 MS-DOS 或 Linux shell）中运行，但大多数计算机用户都希望软件有图形用户界面，可以通过鼠标和键盘进行输入。

对初级开发人员来说，编写窗口软件是一项更具挑战性的任务。但正如上一章介绍的，Java 2 使用 Swing（一组用于创建和使用图形用户界面的类）简化了这项工作。

Swing 提供了如下特性。

- 常见的用户界面组件：按钮、文本框、文本区域、标签、复选框、单选按钮、滚动条、列表、菜单项和滑块。
- 容器：用于放置其他组件的界面组件，包括框架、面板、窗口、菜单、菜单栏和选项卡窗格 (tabbed pane)。
- 可调整的外观：能够修改整个界面的风格，使之类似于 Windows、MacOS 或其他独特的设计。

### 10.1 Swing 的特性

上一章介绍的大多数组件和容器都是 AWT 类的 Swing 版本，AWT 是 Java 中最初的图形用户界面编程包。

Swing 提供了很多全新的特性，包括可定义的外观、键盘记忆、工具提示和标准对话框。

#### 10.1.1 设置外观

在 Swing 中，不同寻常的特性之一能够定义组件的外观，即在屏幕上显示按钮、标签和其他图形用户界面元素的方式。

外观的管理工作是由 javax.swing 包中的用户界面管理器类 UIManager 来完成的。可供选择的外观随 Java 开发环境而异。下面是在 Windows XP 平台上，可用的 Java 外观：

- Windows 外观；
- Windows Classic 外观；
- Motif X Window 外观；
- Metal 外观，这是 Swing 的跨平台 Java 外观。

图 10.1、图 10.2 和图 10.3 显示了同一个图形用户界面的不同外观 (Metal、Windows Classic 和 Motif)。

**注意**

图 10.1~图 10.3 所示的图形用户界面是用本周介绍的技术（其中一些还未介绍）创建的。

类 `UIManager` 有一个 `setLookAndFeel(LookAndFeel)` 方法，可用于选择程序的外观。要得到一个能用于 `setLookAndFeel()` 中的 `LookAndFeel` 对象，可以使用 `UIManager` 的下列方法。

- `getCrossPlatformLookAndFeelClassName()`：返回一个表示 Java 跨平台外观（Ocean）的 `LookAndFeel` 对象。
- `getSystemLookAndFeelClassName()`：返回一个表示当前系统外观的 `LookAndFeel` 对象。

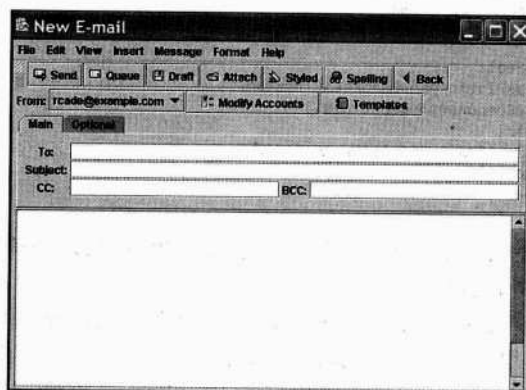


图 10.1 Java 外观 (Metal)

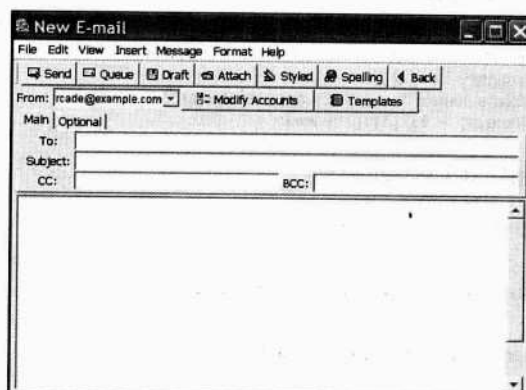


图 10.2 Windows Classic 外观

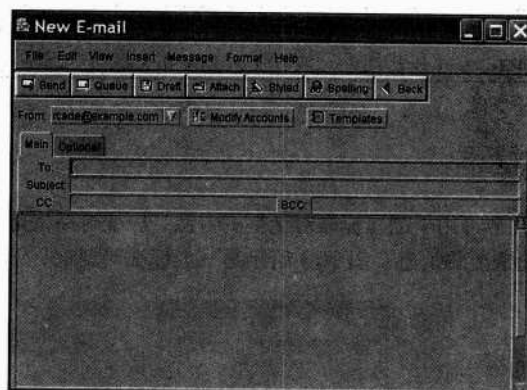


图 10.3 Motif 外观

如果指定的外观无法设置, `setLookAndFeel()` 将引发 `UnsupportedLookAndFeelException` 异常。

调用该方法后, 必须命令界面中的每个组件更新外观。为此, 可以调用 `SwingUtilities` 类的 `updateComponentTreeUI(Component)` 方法, 并将主界面组件 (如 `JFrame` 对象) 作为参数传递给它。

在大多数情况下, 应在每个组件都被加入到图形用户界面后 (即让界面可见之前) 调用 `setLookAndFeel()` 方法。

下面的语句将组件设置为 Java 外观:

```
try {
    UIManager.setLookAndFeel(
        UIManager.getCrossPlatformLookAndFeelClassName());
    SwingUtilities.updateComponentTreeUI(this);
} catch (Exception e) {
    System.out.println("Can't set look and feel: " +
        e.getMessage());
    e.printStackTrace();
}
```

关键字 `this` 指的是包含上述语句的类。如果将这些语句放在 `JFrame` 的构造方法的最后, 则该框架中的所有组件都将以 Java 外观显示。

要选择系统外观, 可使用方法 `getSystemLookAndFeelClassName()`, 它位于上述示例的 `setLookAndFeel()` 方法调用中。这在不同的操作系统上将产生不同的结果: 通过调用 `getSystemLookAndFeelClassName()`, Windows 用户将得到 Windows 外观; UNIX 用户将得到 Motif 外观; 而 Mac X 用户将得到 Aque 外观。

如果不知道操作系统中有哪些外观设计, 可使用下面的语句来列出它们:

```
UIManager.LookAndFeelInfo[] laf = UIManager.getInstalledLookAndFeels();
for (int i = 0; i < laf.length; i++) {
    System.out.println("Class name: " + laf[i].getClassName());
    System.out.println("Name: " + laf[i].getName() + "\n");
}
```

在 Windows 中, 上述语句将生成如下输出:

```
Name: Metal
Class name: javax.swing.plaf.metal.MetalLookAndFeel

Name: CDE/Motif
Class name: com.sun.java.swing.plaf.motif.MotifLookAndFeel

Name: Windows
Class name: com.sun.java.swing.plaf.windows.WindowsLookAndFeel

Name: Windows Classic
Class name: com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel.
```

#### 警告

由于版权方面的原因, Windows 和 MacOS 外观都不能在运行非相应操作系统的机器上使用。在 Windows 计算机中, 不能使用 Mac 外观, 反之亦然。

### 10.1.2 标准对话框

`JOptionPane` 类提供了几个可用于创建标准对话框的方法, 标准对话框是一个小窗口, 用于询问问题、警告用户或提供简短而重要的消息。图 10.4 所示是一个标准对话框。

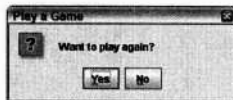


图 10.4 标准对话框

图 10.4 以及本章其他的示例使用 Metal 外观，它是一种跨平台外观，是 Java 软件的默认外观。

毫无疑问，您见到过这样的对话框。当系统崩溃时，将出现一个对话框，并发布这种坏消息。当您删除文件时，将出现一个对话框，确认您是否真的要这样做。

这些窗口是与用户交流的有效途径，您无需创建新的类来表示这些窗口，不用添加组件，也不用编写事件处理方法来接收输入。当使用 `JOptionPane` 提供的标准对话框时，所有这些任务都将自动完成。

4 个标准对话框分别如下。

- 确认对话框：询问问题，包含用于 Yes、No 和 Cancel 响应的按钮。
- 输入对话框：提示用户输入文本。
- 消息对话框：显示消息。
- 选项对话框：包含上面 3 种对话框。

在 `JOptionPane` 类中，有对应于每种对话框的方法。

### 1. 确认对话框

要创建 Yes/No/Cancel 对话框，最简单的方式是调用 `showConfirmDialog(Component, Object)` 方法。`Component` 参数指出了包含对话框的容器，用于确定对话框窗口应被显示在屏幕的什么位置。如果为 `null` 或指定的容器不是 `JFrame` 对象，对话框将被显示在屏幕中央。

第二个参数 (`Object`) 可以是字符串、组件或 `Icon` 对象。如果是一个字符串，其中的文本将被显示在对话框中；如果是一个组件或 `Icon`，则将显示该对象（而不是文本信息）。

该方法返回下列 3 个可能的整数值之一，它们都是 `JOptionPane` 中的类常量：`YES_OPTION`、`NO_OPTION` 和 `CANCEL_OPTION`。

下面的例子使用了一个包含一条文本消息的确认对话框，并将响应存储在变量 `response` 中：

```
int response = JOptionPane.showConfirmDialog(null,
    "Should I delete all of your irreplaceable personal files?");
```

图 10.5 显示了该对话框。

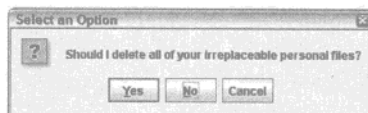


图 10.5 确认对话框

方法 `showConfirmDialog(Component, Object, String, int, int)` 提供了更多的确认对话框选项。前两个参数与 `showConfirmDialog()` 方法相同，后 3 个参数如下：

- 将显示在对话框标题栏中的字符串，
- 一个整数，指出将显示哪些选项按钮，应为类变量 `YES_NO_CANCEL_OPTION` 或 `YES_NO_OPTION`；
- 一个描述对话框类型的整数，值为类变量 `ERROR_MESSAGE`、`INFORMATION_MESSAGE`、`PLAIN_MESSAGE`、`QUESTION_MESSAGE` 或 `WARNING_MESSAGE`。该参数决定了除消息外，对话框中还应有哪种图标。

例如：

```
int response = JOptionPane.showConfirmDialog(null,
    "Error reading file. Want to try again?",
    "File Input Error",
    JOptionPane.YES_NO_OPTION,
    JOptionPane.ERROR_MESSAGE);
```

图 10.6 显示了上述代码生成的对话框。

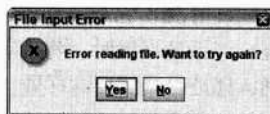


图 10.6 包含按钮 Yes 和 No 的确认对话框

## 2. 输入对话框

输入对话框询问一个问题，并使用文本框来存储用户的响应，如图 10.7 所示。

要创建输入对话框，最简单的方式是调用方法 `showInputDialog(Component, Object)`。其中的参数分别是父组件和对话框中显示的字符串、组件或图标。

该方法返回一个表示用户响应的字符串。下面的语句创建如图 10.7 所示的输入对话框：

```
String response = JOptionPane.showInputDialog(null,
    "Enter your name:");
```

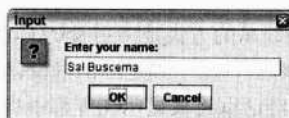


图 10.7 输入对话框

也可以使用方法 `showInputDialog(Component, Object, String, int)` 来创建输入对话框。其中前两个参数与前一个方法相同，后两个参数的含义如下：

- 对话框标题栏上的标题；
- 描述对话框类型的 5 个类常量之一：`ERROR_MESSAGE`、`INFORMATION_MESSAGE`、`PLAIN_MESSAGE`、`QUESTION_MESSAGE` 或 `WARNING_MESSAGE`。

下面的语句使用该方法创建了一个输入对话框：

```
String response = JOptionPane.showInputDialog(null,
    "What is your ZIP code?",
    "Enter ZIP Code",
    JOptionPane.QUESTION_MESSAGE);
```

## 3. 消息对话框

消息对话框是一个显示信息的简单窗口，如图 10.8 所示。

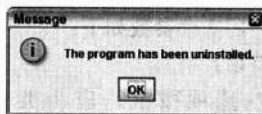


图 10.8 消息对话框

消息对话框可使用方法 `showMessageDialog(Component, Object)` 来创建。与其他对话框一样，这些参数分别是父组件和要显示的字符串、组件或图标。

与其他对话框不同，消息对话框并不返回任何响应值。下面的语句创建了一个如图 10.8 所示的消息对话框：

```
JOptionPane.showMessageDialog(null,
    "The program has been uninstalled.");
```

也可以使用方法 `showMessageDialog(Component, Object, String, int)` 来创建消息对话框，其用法与方法 `showInputDialog()` 完全相同，参数也一样，只是 `showMessageDialog()` 不返回任何值。

下面的语句使用该方法创建了一个消息对话框：

```
JOptionPane.showMessageDialog(null,
    "An asteroid has destroyed the Earth.",
    "Asteroid Destruction Alert",
    JOptionPane.WARNING_MESSAGE);
```

#### 4. 选项对话框

最复杂的对话框是选项对话框，它融其他所有对话框的特性于一体。这种对话框可使用方法 `showOptionDialog(Component, Object, String, int, int, Icon, Object[], Object)` 来创建。

其中各个参数的含义如下：

- 对话框的父组件；
- 要显示的文本、图标或组件；
- 要在标题栏中显示的字符串；
- 对话框类型，值为类常量 `YES_NO_OPTION` 或 `YES_NO_CANCEL_OPTION`，如果要使用其他按钮，则为字面量 `0`；
- 要显示的图标，值为类常量 `ERROR_MESSAGE`、`INFORMATION_MESSAGE`、`PLAIN_MESSAGE`、`QUESTION_MESSAGE` 或 `WARNING_MESSAGE`，如果要使用其他按钮，则为字面量 `0`；
- 要显示的 `Icon` 对象，它将替换前一个参数指定的图标；
- 一个对象数组，其中存储了表示对话框中选项的组件和其他对象，这出现在第 4 个参数的值不为 `YES_NO_OPTION` 或 `YES_NO_CANCEL_OPTION` 时；
- 没有使用 `YES_NO_OPTION` 或 `YES_NO_CANCEL_OPTION` 时，表示默认选项的对象。

最后两个参数让您能够定制对话框。您可以创建一个字符串数组，用于存储对话框中每个按钮上的文本。这些组件是使用顺序布局管理器显示的。

下面的例子创建了一个选项对话框，它使用一个 `String` 对象数组作为对话框中的选项，并将元素 `gender[2]` 用作默认项：

```
String[] gender = { "Male", "Female",
    "None of Your Business" };
int response = JOptionPane.showOptionDialog(null,
    "What is your gender?",
    "Gender",
    0,
    JOptionPane.INFORMATION_MESSAGE,
    null,
    gender,
    gender[2]);
```

图 10.9 显示了上述代码生成的对话框。

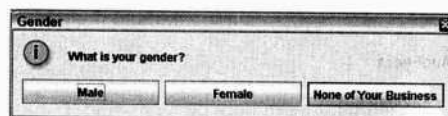


图 10.9 选项对话框

### 10.1.3 使用对话框

下面的应用程序显示了一系列的对话框。应用程序 `FeedInfo` 使用对话框从用户那里获取信息，然后将这些信息放在应用程序主窗口的文本框中。

请输入并编译程序清单 10.1 中的代码。



程序清单 10.1 完整的 FeedInfo.java 源代码

```

1: import java.awt.GridLayout;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class FeedInfo extends JFrame {
6:     private JLabel nameLabel = new JLabel("Name: ",
7:         SwingConstants.RIGHT);
8:     private JTextField name;
9:     private JLabel urlLabel = new JLabel("URL: ",
10:         SwingConstants.RIGHT);
11:     private JTextField url;
12:     private JLabel typeLabel = new JLabel("Type: ",
13:         SwingConstants.RIGHT);
14:     private JTextField type;
15:
16:     public FeedInfo() {
17:         super("Feed Information");
18:         setSize(400, 105);
19:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20:         setLookAndFeel();
21:         // Site name
22:         String response1 = JOptionPane.showInputDialog(null,
23:             "Enter the site name:");
24:         name = new JTextField(response1, 20);
25:
26:         // Site address
27:         String response2 = JOptionPane.showInputDialog(null,
28:             "Enter the site address:");
29:         url = new JTextField(response2, 20);
30:
31:         // Site type
32:         String[] choices = { "Personal", "Commercial", "Unknown" };
33:         int response3 = JOptionPane.showOptionDialog(null,
34:             "What type of site is it?",
35:             "Site Type",
36:             0,
37:             JOptionPane.QUESTION_MESSAGE,
38:             null,
39:             choices,
40:             choices[0]);
41:         type = new JTextField(choices[response3], 20);
42:
43:         setLayout(new GridLayout(3, 2));
44:         add(nameLabel);
45:         add(name);
46:         add(urlLabel);
47:         add(url);
48:         add(typeLabel);
49:         add(type);
50:         setLookAndFeel();
51:         setVisible(true);
52:     }
53:
54:     private void setLookAndFeel() {
55:         try {
56:             UIManager.setLookAndFeel(
57:                 UIManager.getSystemLookAndFeelClassName());
58:             SwingUtilities.updateComponentTreeUI(this);
59:         } catch (Exception e) {
60:             System.err.println("Couldn't use the system "
61:                 + "look and feel: " + e);
62:         }
63:     }
64:
65:     public static void main(String[] arguments) {
66:         FeedInfo frame = new FeedInfo();
67:     }
68: }

```

当您填写好对话框中的文本框后，将看到该应用程序的主窗口，如图 10.10 所示，它具有 Windows

外观。其中的 3 个文本框中的值分别是由 3 个对话框提供的。

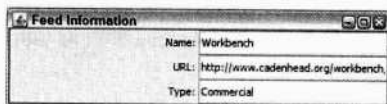


图 10.10 应用程序 FeedInfo 的主窗口

该应用程序中的很多代码都是样板代码，可用于任何 Swing 程序中。下面的这些代码行与对话框有关。

- 第 22~24 行：一个要求用户输入网站名称的输入对话框。该名称被构造函数用来创建一个 JTextField 对象，它将该名称放入文本框中。
- 第 27~29 行：一个询问网站地址的输入对话框，它被构造函数用来创建另一个 JTextField 对象。图 10.10 显示了该对话框。
- 第 32 行：创建一个名为 choices 的 String 对象，并给 3 个元素赋值。
- 第 33~40 行：创建一个询问网站类型的选项对话框。数组 choices 被用作第 7 个参数，它用数组中的字符串在对话框上设置了 3 个按钮(Personal, Commercial 和 Unknown)。最后一个参数 choices[0]，指定了对话框的默认选项是第一个按钮。
- 第 41 行：将选项对话框的响应(一个指出所选数组元素的整数值)存储在名为 type 的 JTextField 组件中。

在框架的构造函数的开头和末尾分别调用了第 54~63 行的 setLookAndFeel() 方法创建的外观。由于在构造函数中将打开多个对话框，因此必须在打开它们之前设置其外观。

### 10.1.4 滑块

在 Swing 中，滑块是使用类 JSlider 来实现的，它让用户能够通过滑动控制来设置一个位于最大值和最小值之间的值。在很多情况下，都可以使用滑块(而不是文本框)来获取数字输入，其优点是能够限制可输入的值。

图 10.11 显示的是一个 JSlider 组件。

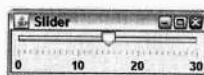


图 10.11 JSlider 组件

默认情况下，滑块是水平方向的，可以使用接口 SwingConstants 的两个类常量(HORIZONTAL 和 VERTICAL)来显式地设置其方向。

可以使用下面的构造方法。

- JSlider(int, int): 一个具有指定的最小和最大值的滑块。
- JSlider(int, int, int): 一个具有指定的最小和最大值以及初始值的滑块。
- JSlider(int, int, int, int): 一个具有指定的方向、最小值、最大值和初始值的滑块。

滑块有一个可选的标签，可用于指出最小值、最大值以及显示两组不同的刻度线。默认情况下，最小值为 0，最大值为 100，初始值为 50，方向为水平。

标签的各元素是通过调用如下几个 JSlider 方法创建的。

- setMajorTickSpacing(int): 按指定的间隔放置主刻度线。间隔不是以像素为单位的，而是用滑块表示的最大和最小值之间的值来设定。

- `setMinorTickSpacing(int)`: 按指定的间隔放置次刻度线, 次刻度线高度只有主刻度线的一半。
- `setPaintTicks(boolean)`: 决定是否显示刻度线 (参数为 `true` 时显示; 参数为 `false` 时不显示)。
- `setPaintLabels(boolean)`: 决定是否显示数字标签 (参数为 `true` 时显示; 参数为 `false` 时不显示)。

应在将滑块加入到容器之前调用这些方法。

程序清单 10.2 是 `Slider.java` 的源代码。

程序清单 10.2 完整的 `Slider.java` 源代码

```
1: import java.awt.event.*;
2: import javax.swing.*;
3:
4: public class Slider extends JFrame {
5:
6:     public Slider() {
7:         super("Slider");
8:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9:         JSlider pickNum = new JSlider(JSlider.HORIZONTAL, 0, 30, 5);
10:        pickNum.setMajorTickSpacing(10);
11:        pickNum.setMinorTickSpacing(1);
12:        pickNum.setPaintTicks(true);
13:        pickNum.setPaintLabels(true);
14:        add(pickNum);
15:    }
16:
17:    public static void main(String[] args) {
18:        Slider frame = new Slider();
19:        frame.pack();
20:        frame.setVisible(true);
21:    }
22: }
```

第 9~14 行包含用于创建 `JSlider` 组件, 设置要显示其刻度线, 并将它加入到容器中的代码。程序的余下部分是基本的应用程序框架, 它由一个不带菜单的主 `JFrame` 容器构成。

第 18~20 行创建一个新的 `Slider` 对象, 并调用该对象的 `pack()` 方法将大小设置为刚好显示其中的组件, 然后显示该对象。

#### 注意

在框架的构造函数外调用 `pack()` 和 `setVisible()` 方法好像有些奇怪。这些方法是公有的, 因此在界面组件类的外面还是里面调用这些方法 (和其他方法), 并没有任何区别。

### 10.1.5 滚动窗格

正如前一章介绍的, 在以前的 Java 版本中, 一些组件 (如文本区域) 内置了滚动条。当组件无法将其中的文本一次性全部显示出来时, 可以使用滚动条。滚动条可以是垂直或水平的。

最为常见的滚动示例是, 在 Web 浏览器中, 可以在任何大于浏览器显示区域的页面中使用滚动条。Swing 将滚动条规则修改为:

- 要让组件能够滚动, 必须将它加入到 `JScrollPane` 容器中。
- 该 `JScrollPane` 容器 (而不是可滚动的组件) 将被加入到其他容器中。

可使用构造函数 `ScrollPane(Object)` 来创建滚动窗格, 其中 `Object` 表示能被滚动的组件。

下面的例子在滚动窗格 (scroller) 中创建了一个文本区域, 并将该滚动窗格加入到名为 `mainPane` 的容器中:

```
textBox = new JTextArea(7, 30);
JScrollPane scroller = new JScrollPane(textBox);
mainPane.add(scroller);
```

使用滚动窗格时，指定它在界面中的大小通常很有用。为此，可以在将它加入到容器中之前，调用其方法 `setPreferredSize(Dimension)`。对象 `Dimension` 表示了宽度和高度（单位为像素）。

下面的代码在前一个例子的基础上，设置了对象 `scroller` 的大小：

```
Dimension pref = new Dimension(350, 100);
scroller.setPreferredSize(pref);
```

上述操作应在将对象 `scroller` 加入到容器中之前完成。

#### 警告

要使 Swing 正确运行，在很多情况下，必须按正确的顺序完成某些工作，前面介绍的就是这样一种情况。对于大多数组件而言，顺序如下：创建组件、设置组件，然后将组件加入到容器中。

默认情况下，滚动窗格仅在需要时才会显示滚动条。如果窗格中的组件比窗格本身小，滚动条将不会出现。对于诸如文本区域等组件，组件的大小可能随着程序的运行而增大，在需要时，滚动条将自动出现，而不需要时，滚动条又将自动消失。

要覆盖这种行为，可在创建 `JScrollBar` 组件时，使用下面 `ScrollPaneConstants` 类常量之一设置其策略 (policy)：

- `HORIZONTAL_SCROLLBAR_ALWAYS`;
- `HORIZONTAL_SCROLLBAR_AS_NEEDED`;
- `HORIZONTAL_SCROLLBAR_NEVER`;
- `VERTICAL_SCROLLBAR_ALWAYS`;
- `VERTICAL_SCROLLBAR_AS_NEEDED`;
- `VERTICAL_SCROLLBAR_NEVER`。

这些类常量用于构造函数 `ScrollPane(Object, int, int)` 中，后者指定了窗格中的组件、垂直滚动条的策略和水平滚动条的策略。

#### 注意

对于任何需要滚动的 Swing 组件，都可将其加入到滚动窗格中。如果要滚动文本区域，并在用户添加新文本时跳到窗格底部，可调用文本区域组件的方法 `setCreatePosition(getDocument().getLength())`。`setCreatePosition` 的上述参数指出了文本区域当前包含多少文本。

### 10.1.6 工具栏

在 Swing 中，使用 `JToolBar` 类来创建工具栏，后者是一个容器，它将多个组件组织为一行或一列。被组织的组件通常是按钮。

工具栏用于将最常用的程序选项组织在一起。工具栏中通常包含按钮和列表，可用于替代下拉菜单和快捷键。

默认情况下，工具栏是水平的，但可以使用接口 `SwingConstants` 中的类常量 `HORIZONTAL` 或 `VERTICAL` 来显式地设置其方向。

构造方法包括：

- `JToolBar()`，新建一个工具栏；
- `JToolBar(int)`，创建了工具栏，并指定其方向。

创建工具栏后，可以使用其方法 `add(Object)` 来加入其他组件，其中 `Object` 是要加入到工具栏中的组件。

很多使用了工具栏的程序都允许用户移动工具栏。这被称为可停放工具栏 (dockable toolbar)，因

为可以将它们停放在屏幕边缘，这类似于将船停靠到码头。Swing 的工具栏也可停放到新窗口内，从而与原来的窗口分离。

为获得最佳结果，应使用 BorderLayout 管理器将可停放的 JToolBar 组件放置到容器中。边界布局将容器分为 5 个区域：北、南、东、西、中，其中每个有方向的组件都占据了它所需的空間，剩下的被分配给中央区域。

工具栏应该被放置到边界布局的方向区域中。布局中唯一能被填充的是中央区域（有关诸如边界布局等布局管理器的更详细信息，将在下一章介绍）。

图 10.12 是一个可停放的工具栏，它占据了边界布局的北方区域，而中央是一个文本区域。

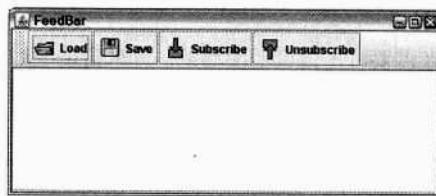


图 10.12 可停放的工具栏和文本区域

程序清单 10.3 是该应用程序的源代码。

#### 程序清单 10.3 完整的 FeedBar.java 源代码

```

1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class FeedBar extends JFrame {
6:
7:     public FeedBar() {
8:         super("FeedBar");
9:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10:        // create icons
11:        ImageIcon loadIcon = new ImageIcon("load.gif");
12:        ImageIcon saveIcon = new ImageIcon("save.gif");
13:        ImageIcon subscribeIcon = new ImageIcon("subscribe.gif");
14:        ImageIcon unsubscribeIcon = new ImageIcon("unsubscribe.gif");
15:        // create buttons
16:        JButton load = new JButton("Load", loadIcon);
17:        JButton save = new JButton("Save", saveIcon);
18:        JButton subscribe = new JButton("Subscribe", subscribeIcon);
19:        JButton unsubscribe = new JButton("Unsubscribe", unsubscribeIcon);
20:        // add buttons to toolbar
21:        JToolBar bar = new JToolBar();
22:        bar.add(load);
23:        bar.add(save);
24:        bar.add(subscribe);
25:        bar.add(unsubscribe);
26:        // prepare user interface
27:        JTextArea edit = new JTextArea(8, 40);
28:        JScrollPane scroll = new JScrollPane(edit);
29:        BorderLayout bord = new BorderLayout();
30:        setLayout(bord);
31:        add("North", bar);
32:        add("Center", scroll);
33:        pack();
34:        setVisible(true);
35:    }
36:
37:    public static void main(String[] arguments) {
38:        FeedBar frame = new FeedBar();
39:    }
40: }

```

该应用程序使用 4 个图像来表示按钮上的图形。这些图形与前一章中 IconFrame 项目使用的相同，如果您还没有下载它们，您也可以使用自己系统上的图形，但必须是 GIF 格式，并且很小。

可通过手柄（图 10.12 中 Load 按钮左边的区域）来拖动该应用程序中的工具栏。如果在窗口内拖动它，可将它停放在应用程序窗口的边界上。当放开该工具栏时，应用程序将使用边界布局管理器重新排列其中的组件。您还可以将工具栏完全拖出应用程序窗口。

虽然在通常情况下，工具栏包含的是图形按钮，但也可以包含文本按钮、组合框以及其他组件。

### 10.1.7 进度条

进度条（progress bar）用于显示用户在任务完成前还需要等待多长时间。

在 Swing 中，进度条是使用类 JProgressBar 来实现的。图 10.13 显示了一个使用这种组件的 Java 程序。

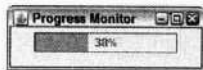


图 10.13 框架中的进度条

进度条用于跟踪可用数字表示的任务进度。通过指定最小和最大值来创建它们，这些值表示任务的起点和终点。

一个可用数字表示任务进度的例子是，安装一个由 335 个不同文件组成的软件。可使用传输的文件数来监视任务的进度，其中最小值为 0，最大值为 335。

构造方法包括：

- JProgressBar(), 创建一个新的进度条；
- JProgressBar(int, int), 创建一个有指定的最小值和最大值的进度条；
- JProgressBar(int, int, int), 创建一个有指定的方向、最小值和最大值的进度条。

可以使用类常量 SwingConstants.VERTICAL 和 SwingConstants.HORIZONTAL 来设置进度条的方向。默认情况下，进度条的方向是水平的。

最小值和最大值可以通过调用进度条的 setMinimum(int)和 setMaximum(int)来设置。

要更新进度条，可调用它的 setValue(int)方法，并将指定一个当前进度的值传递给它，这个值应该位于进度条的最小值和最大值之间。下面的例子告诉进度条 install，前面的软件安装示例已上传了多少个文件。

```
int filesDone = getNumberOfFiles();
install.setValue(filesDone);
```

其中，方法 getNumberOfFiles()表示用于跟踪已复制的文件个数的代码。当这个值由方法 setValue()传递给进度条时，进度条将被立即更新，以指出已完成的比例。

除了包含空的、待填充的框外，进度条通常还包括一个文本标签。该标签显示了已经完成的任务的百分比，可调用方法 setStringPainted(boolean)并将 true 作为参数来显示它，如果使用参数 false，将不显示该标签。

程序清单 10.4 是应用程序 ProgressMonitor，其运行情况如图 10.14 所示。

#### 程序清单 10.4 完整的 ProgressMonitor.java 源代码

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
```

```

5: public class ProgressMonitor extends JFrame {
6:
7:     JProgressBar current;
8:     JTextArea out;
9:     JButton find;
10:    Thread runner;
11:    int num = 0;
12:
13:    public ProgressMonitor() {
14:        super("Progress Monitor");
15:
16:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17:        setSize(205, 68);
18:        setLayout(new FlowLayout());
19:        current = new JProgressBar(0, 2000);
20:        current.setValue(0);
21:        current.setStringPainted(true);
22:        add(current);
23:    }
24:
25:
26:    public void iterate() {
27:        while (num < 2000) {
28:            current.setValue(num);
29:            try {
30:                Thread.sleep(1000);
31:            } catch (InterruptedException e) { }
32:            num += 95;
33:        }
34:    }
35:
36:    public static void main(String[] arguments) {
37:        ProgressMonitor frame = new ProgressMonitor();
38:        frame.setVisible(true);
39:        frame.iterate();
40:    }
41: }

```

应用程序 ProgressMonitor 使用进度条来跟踪变量 num 的值，该进度条是在第 18 行创建的，其最小值为 0，最大值为 2 000。

第 26~34 行的 iterate() 方法在 num 小于 2 000 时不断进行循环，每次将 num 加 95。第 27 行调用进度条的 setValue() 方法，并将 num 作为参数传递给它，使进度条使用这个值来显示进度。

当程序将执行一段时间时，使用进度条可使程序对用户更友好。软件用户喜欢进度条，因为它指出了完成某项工作大概还需多长时间。

进度条还提供了另一种重要的信息：表明程序还在运行，而没有崩溃。

### 10.1.8 菜单

提高框架的可用性的方法之一是添加一个菜单栏——一系列用于执行任务的下拉菜单。菜单的功能通常也可用按钮和其他用户界面组件来完成，这样用户可以通过两种不同的方式来完成工作。

在 Java 中，3 种组件协同工作来支持菜单。

- JMenuItem：菜单中的一个菜单项。
- JMenu：一个下拉菜单，包含一个或多个 JMenuItem 组件、其他界面组件和分隔符（位于菜单项之间的线段）。
- JMenuBar：包含一个或多个 JMenu 组件并显示其名称的容器。

JMenuItem 类似于按钮，其构造函数与 JButton 组件类似。要创建文本菜单项，可调用 JMenuItem(String)；要创建显示图形文件的菜单项，可调用 JMenuItem(Icon)；要创建包含文本和图标的菜单项，可调用 JMenuItem(String, Icon)。



下面的语句创建了 7 个菜单项：

```
JMenuItem j1 = new JMenuItem("Open");
JMenuItem j2 = new JMenuItem("Save");
JMenuItem j3 = new JMenuItem("Save as Template");
JMenuItem j4 = new JMenuItem("Page Setup");
JMenuItem j5 = new JMenuItem("Print");
JMenuItem j6 = new JMenuItem("Use as Default Message Style");
JMenuItem j7 = new JMenuItem("Close");
```

JMenu 容器存放了一个下拉菜单中的所有菜单项。要创建这种容器，可调用构造函数 JMenu(String)，并将菜单名作为参数传递给它，该名称将被显示在菜单栏中。

创建 JMenu 容器后，调用其 add(JMenuItem) 方法将菜单项加入。新加入到的菜单项将被放置在菜单的最后。

也可以将其他组件加入到菜单中，方法是调用 add(Component) 方法，并将要加入的用户界面组件作为参数传递给它。常出现在菜单中的组件之一是复选框（在 Java 中，是 JCheckBox 类）。

要将分隔符加入到菜单末尾，可调用 addSeparator() 方法。分隔符通常用于将菜单中多个相关的菜单项分成一组。

也可以将文本加入到菜单中，以用作标签。为此，可以调用 add(String) 方法，并将要加入的文本作为参数传递给它。

下面的语句创建了一个菜单，并将前面创建的 7 个菜单项以及 3 个分隔符加入到其中：

```
JMenu m1 = new JMenu("File");
m1.add(j1);
m1.add(j2);
m1.add(j3);
m1.addSeparator();
m1.add(j4);
m1.add(j5);
m1.addSeparator();
m1.add(j6);
m1.addSeparator();
m1.add(j7);
```

JMenuBar 容器包含一个或多个 JMenu 容器并显示它们的名称。菜单栏通常位于应用程序标题栏的下面。

要创建菜单栏，可调用构造方法 JMenuBar()，且不提供任何参数。然后，可以调用其 add(JMenu) 方法，将菜单加入到菜单栏的最后面。

创建所有的菜单项、将其加入到菜单中，并将菜单加入到菜单栏中后，便可以将菜单栏加入到框架中。为此，可以调用框架的 setJMenuBar(JMenuBar) 方法。

下面的语句创建了一个菜单栏，将菜单加入其中，将菜单栏加入到名为 gui 的框架中：

```
JMenuBar bar = new JMenuBar();
bar.add(m7);
gui.setJMenuBar(bar);
```

图 10.14 显示了该菜单栏在一个本来为空的框架中的情况。

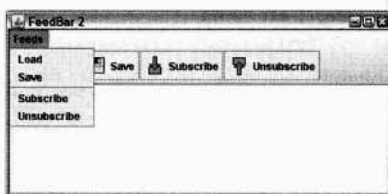


图 10.14 包含菜单栏的框架

虽然您可以打开、关闭该菜单并选择其中的菜单项，但这样做，系统并不会做出任何响应。第 12 章将介绍如何让这种组件（以及其他组件）接收用户的输入。



程序清单 10.5 是 FeedBar 项目的扩展版本。它新增了一个菜单栏，其中包含 1 个菜单和 4 个菜单项。该应用程序的运行情况如图 10.14 所示。

程序清单 10.5 完整的 FeedBar2.java 源代码

```

1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class FeedBar2 extends JFrame {
6:
7:     public FeedBar2() {
8:         super("FeedBar 2");
9:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10:        // create icons
11:        ImageIcon loadIcon = new ImageIcon("load.gif");
12:        ImageIcon saveIcon = new ImageIcon("save.gif");
13:        ImageIcon subscribeIcon = new ImageIcon("subscribe.gif");
14:        ImageIcon unsubscribeIcon = new ImageIcon("unsubscribe.gif");
15:        // create buttons
16:        JButton load = new JButton("Load", loadIcon);
17:        JButton save = new JButton("Save", saveIcon);
18:        JButton subscribe = new JButton("Subscribe", subscribeIcon);
19:        JButton unsubscribe = new JButton("Unsubscribe", unsubscribeIcon);
20:        // add buttons to toolbar
21:        JToolBar bar = new JToolBar();
22:        bar.add(load);
23:        bar.add(save);
24:        bar.add(subscribe);
25:        bar.add(unsubscribe);
26:        // create menu
27:        JMenuItem j1 = new JMenuItem("Load");
28:        JMenuItem j2 = new JMenuItem("Save");
29:        JMenuItem j3 = new JMenuItem("Subscribe");
30:        JMenuItem j4 = new JMenuItem("Unsubscribe");
31:        JMenuBar menubar = new JMenuBar();
32:        JMenu menu = new JMenu("Feeds");
33:        menu.add(j1);
34:        menu.add(j2);
35:        menu.addSeparator();
36:        menu.add(j3);
37:        menu.add(j4);
38:        menubar.add(menu);
39:        // prepare user interface
40:        JTextArea edit = new JTextArea(8, 40);
41:        JScrollPane scroll = new JScrollPane(edit);
42:        BorderLayout bord = new BorderLayout();
43:        setLayout(bord);
44:        add("North", bar);
45:        add("Center", scroll);
46:        setJMenuBar(menubar);
47:        pack();
48:        setVisible(true);
49:    }
50:
51:    public static void main(String[] arguments) {
52:        FeedBar2 frame = new FeedBar2();
53:    }
54: }

```

### 10.1.9 选项卡窗格

选项卡窗格 (tabbed pane) 是一组堆叠在一起的面板，用户每次只能查看其中的一个，在 Swing 中，这是由 JTabbedPane 类实现的。

要查看其中的面板，用户必须单击包含其名称的标签 (tab)。标签可跨越组件顶部或底部水平排列，也可沿组件左边或右边垂直排列。

用于创建选项卡窗格的构造方法如下：

- `JTabbedPane()`，创建一个选项卡窗格，但不能滚动；
- `JTabbedPane(int)`，创建一个不能滚动的选项卡窗格，且有指定的布局（placement）；
- `JTabbedPane(int, int)`，创建一个有指定布局（第一个参数）和滚动策略（第二个参数）的选项卡窗格。

选项卡窗格的布局（placement）指的是其标签（tab）相对于面板的位置。可使用 4 个类变量之一作为构造函数的参数：`JTabbedPane.TOP`、`JTabbedPane.BOTTOM`、`JTabbedPane.LEFT` 或 `JTabbedPane.RIGHT`。

滚动策略确定当界面无法容纳全部的标签时，标签将如何被显示。不滚动的选项卡窗格将显示多余的标签，这可以使用类变量 `JTabbedPane.WRAP_TAB_LAYOUT` 来设置；滚动的选项卡窗格将在标签旁边显示滚动箭头，这可以使用类变量 `JTabbedPane.SCROLL_TAB_LAYOUT` 来设置。

创建选项卡窗格后，可以调用其 `addTab(String, Component)` 方法来加入组件。其中，`String` 参数将用作标签中显示的文本；第二个参数是一个组件，它将是窗格中的选项卡之一，通常这是一个 `JPanel` 对象，但不一定非得这样。

下面的语句创建了 5 个面板，并将它们加入到一个选项卡窗格中：

```
JPanel mainSettings = new JPanel();
JPanel advancedSettings = new JPanel();
JPanel privacySettings = new JPanel();
JPanel emailSettings = new JPanel();
JPanel securitySettings = new JPanel();
JTabbedPane tabs = new JTabbedPane();
tabs.addTab("Main", mainSettings);
tabs.addTab("Advanced", advancedSettings);
tabs.addTab("Privacy", privacySettings);
tabs.addTab("E-mail", emailSettings);
tabs.addTab("Security", securitySettings);
```

将所有的面板和其他组件都加入到选项卡窗格中后，需要将后者加入到另一个容器中。图 10.15 显示了上述选项卡窗格被加入到框架中的情况。

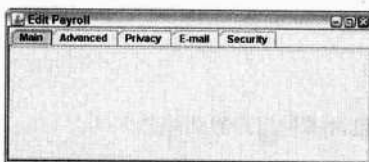


图 10.15 包含 5 个沿顶端水平堆叠的选项卡的窗格

## 10.2 总结

至此，您知道了如何使用 `Swing` 包中的组件在 `Java` 应用程序窗口上绘制用户图形界面。

`Swing` 包含了可用于实现按钮、工具栏、列表、文本框以及高级的组件（如滑块、对话框和进度条等）的类。界面组件是通过创建其所属类的实例，并使用容器的 `add()` 方法或其他类似的方法（如选项卡窗格的 `addTab()` 方法）将其加入到诸如框架等容器中来实现。

在本章中，您开发了一些组件并将它们加入到界面中。接下来的两章将介绍使图形界面更有用的知识：如何排列组件，使之形成一个完整的界面；如何通过组件接收来自用户的输入。

## 10.3 问与答

问：不使用 `Swing`，也能创建应用程序？

答：当然。`Swing` 只是对 `AWT` 的扩展，如果使用较老版本的 `Java` 开发小程序，将只能使用 `AWT`

类来设计界面和接收来自用户的输入。至于创建应用程序时是否应使用 Swing 则是另一个问题。Swing 的功能与 AWT 所提供的功能没有什么可比性。使用 Swing，可以使用更多的组件，并通过更加复杂的方法来控制它们，提高性能和可靠性。

其他用户界面库扩展了 Swing 或与之竞争。其中最流行的一个是 SWT，它是 Eclipse 项目中开发的一个开源图形用户界面库。SWT 提供的组件的外观和行为与操作系统提供的类似。更详细的信息，请访问网站 <http://www.eclipse.org/swt>。

问：在应用程序 Silder 中，`pack()` 语句有何用途？

答：每个界面组件都有最佳的大小，虽然用于在容器中排列组件的布局管理器不会理会这一点。调用框架或窗口的 `pack()` 方法将调用其大小，使之刚好容纳其包含的组件。由于应用程序 Slider 没有设置框架的大小，因此在显示该框架之前应调用 `pack()` 方法，将其设置为适当的大小。

问：我创建选项卡窗格时，显示的只有选项卡，而面板本身不可见。该如何解决这种问题？

答：只有设置好窗格的内容后，选项卡窗格才能正常运行。如果选项卡的窗格为空，则选项卡下面或旁边将不会有任何东西。应确保被加入到选项卡中的面板显示了其所有组件。

## 10.4 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 10.4.1 问题

1. 在 Java 应用程序中，默认的外观是什么？
  - a. Motif;
  - b. Windows;
  - c. Metal。
2. 在软件安装程序中，通常包含哪种用户界面组件？
  - a. 滑块;
  - b. 进度条;
  - c. 对话框。
3. 哪个 Java 类库包含了用于创建可单击按钮的类？
  - a. AWT;
  - b. Swing;
  - c. 两者都有。

答案

1. c，要使用 Metal 之外的外观，必须使用类 `javax.swing.UIManager` 中的方法来显式指定。
2. b，显示文件复制或解压缩的进度时，进度条很有用。
3. c，Swing 包含 AWT 中的所有简单用户界面组件。

### 10.4.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器进行试验。

对于下述代码：

```
import java.awt.*;
import javax.swing.*;

public class AskFrame extends JFrame {
    public AskFrame() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JSlider value = new JSlider(0, 255, 100);
        add(value);
        setSize(450, 150);
        setVisible(true);
        super();
    }

    public static void main(String[] arguments) {
        AskFrame af = new AskFrame();
    }
}
```

如果编译并运行它，将出现什么情况？

- a. 能够通过编译并正确运行；
- b. 能够通过编译，但不会在框架中显示任何东西；
- c. 由于其中的 `super()` 语句而无法通过编译；
- d. 由于其中的 `add()` 语句而无法通过编译。

答案 c

## 10.5 练习

为巩固本章介绍的知识，请尝试完成下面的练习：

- 1. 创建一个输入对话框，用于设置加载该对话框的框架的标题。
- 2. 对应用程序 `ProgressMonitor` 进行修改，使之同时在一个文本框中显示变量 `num` 的值。



## 第 11 章

# 在用户界面上排列组件

如果将设计图形用户界面比作绘画艺术的话，您现在只完成了这门艺术中的一步：抽象表现主义。您可以将组件放到界面上，但不能很好地控制它们的位置。

在 Java 中，要设置界面的格式，必须使用一组被称为布局管理器（layout manager）的类。

本章将介绍如何用 5 个布局管理器来排列界面上的组件。您将利用 Swing 的灵活性，Swing 可用于很多支持 Java 的平台中。

还将介绍当一种布局无法满足程序的需要时，如何将多个布局管理器用于同一个界面。

### 11.1 基本的界面布局

正如上一章中介绍的，使用 Swing 设计的图形用户界面是易变的。调整窗口的大小将破坏界面，因为组件可能在容器中移动，偏离您的期望。

为支持多种平台，这种易变性是必不可少的，因为每个平台显示诸如按钮、滚动条等组件的方式存在细微的差别。

在诸如 Microsoft Visual Basic 等编程语言中，组件在窗口上的位置是由其 (x, y) 坐标准义的。有些 Java 开发工具允许通过使用其窗口类，以类似的方式对界面进行控制（在 Java 中，能够这样做）。

使用 Swing 时，程序员必须使用布局管理器来更好地控制界面布局。

#### 11.1.1 布置界面

布局管理器决定了组件被加入到容器中时将被如何排列。

面板的默认布局管理器是 FlowLayout 类。这个类按组件被加入到容器的顺序从左向右依次排列。第一行排满后，则从第二行开始，继续按从左到右的顺序排列。

Java 提供了一系列通用的布局管理器：FlowLayout、GridLayout、BorderLayout、CardLayout 和 GridBagLayout。要创建布局管理器，首先需要调用布局管理器的构造函数，如下例所示：

```
FlowLayout flo = new FlowLayout();
```

创建布局管理器后，可以使用容器的方法 `setLayout()` 将其与容器关联起来。将组件加入到容器中之前，必须建立容器的布局管理器。如果没有指定布局管理器，将使用默认布局：对于面板，为 FlowLayout；对于框架和窗口，为 BorderLayout。

下面的语句位于框架的开头，它使用一个布局管理器来控制被加入到窗口中的组件的排列方法。

```
import java.awt.*;

public class Starter extends javax.swing.JFrame {

    public Starter() {
        FlowLayout lm = new FlowLayout();
```

```

        setLayout(lm);
        // add components here
    }
}

```

设置布局管理器后，便可以向它管理的容器中加入组件。对诸如 `FlowLayout` 等布局管理器而言，组件的加入顺序非常重要。有关这方面的内容，将在后面讨论各种布局管理器时介绍。

### 11.1.2 顺序布局

类 `FlowLayout` 是最简单的布局管理器，它位于 `java.awt` 包中。它排列组件的方式与图书排版类似：从左到右排列，到达行尾后，进入下一行。

默认情况下，如果调用构造函数 `FlowLayout()` 时没有提供任何参数，每行的组件将居中排列。要让组件右对齐或左对齐，必须将类变量 `FlowLayout.LEFT` 或 `FlowLayout.RIGHT` 作为唯一的参数传递给构造函数，如下面的语句所示：

```

FlowLayout righty = new FlowLayout(FlowLayout.RIGHT);
类变量 FlowLayout.CENTER 用于将组件居中。

```

#### 注意

针对非英语用户排列组件——从左到右排列不合适时，可使用变量 `FlowLayout.LEADING` 和 `FlowLayout.TRAILING`，它们分别让第一个和最后一个组件对齐。

程序清单 11.1 中的应用程序使用顺序布局管理器排列了 6 个按钮。由于调用构造函数 `FlowLayout()` 时将类变量 `FlowLayout.LEFT` 作为参数，因此这些组件从应用程序窗口的左边界开始排列。

程序清单 11.1 完整的 `Alphabet.java` 源代码

```

1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class Alphabet extends JFrame {
6:     JButton a = new JButton("Alibi");
7:     JButton b = new JButton("Burglar");
8:     JButton c = new JButton("Corpse");
9:     JButton d = new JButton("Deadbeat");
10:    JButton e = new JButton("Evidence");
11:    JButton f = new JButton("Fugitive");
12:
13:    public Alphabet() {
14:        super("Alphabet");
15:        setSize(360, 120);
16:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17:        FlowLayout lm = new FlowLayout(FlowLayout.LEFT);
18:        setLayout(lm);
19:        add(a);
20:        add(b);
21:        add(c);
22:        add(d);
23:        add(e);
24:        add(f);
25:        setVisible(true);
26:    }
27:
28:    public static void main(String[] arguments) {
29:        Alphabet frame = new Alphabet();
30:    }
31: }

```

图 11.1 显示了该应用程序运行的情况。

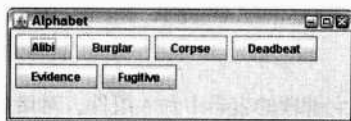


图 11.1 以顺序布局的方式排列 6 个按钮

在应用程序 Alphabet 中, 顺序布局管理器使用默认的水平间距 (5 个像素) 和垂直间距 (5 个像素)。您可以在调用构造函数 `FlowLayout()` 时, 提供其他的参数来修改水平间距和垂直间距。

构造函数 `FlowLayout(int, int, int)` 接受如下 3 个参数。

- 对齐方式: 必须是 `FlowLayout` 的 5 个类变量之一——`.CENTER`、`.LEFT`、`.RIGHT`、`.LEADING` 或 `.TRAILING`。

- 组件间的水平间距, 单位为像素。

- 组件间的垂直间距, 单位为像素。

下面的构造函数创建了一个顺序布局管理器, 它将组件居中, 组件间的水平间距为 30 像素, 垂直间距为 10 像素。

```
FlowLayout flo = new FlowLayout(FlowLayout.CENTER, 30, 10);
```

### 11.1.3 方框布局

方框布局 (box layout) 管理器将组件从左到右或从上到下排列, 它是由 `javax.swing` 包中的 `BoxLayout` 类表示的, 对顺序布局做了改进——不管容器的大小如何变化, 组件总是排列成一行或一列。

使用构造函数创建方框布局管理器时, 必须提供两个参数: 它将管理的容器和指定水平还是垂直排列的类变量。

排列方式是使用 `BoxLayout` 的类变量指定的: `X_AXIS` 表示按从左到右的顺序水平排列; `Y_AXIS` 表示按从上到下的顺序垂直排列。

下面的代码指定面板采用垂直方框布局:

```
JPanel optionPane = new JPanel();
BoxLayout box = new BoxLayout(optionPane,
    BoxLayout.Y_AXIS);
```

被加入到容器中的组件将按指定的方式排列, 并以最合适的大小显示。水平排列时, 方框布局管理器将试图让每个组件的高度相同; 垂直排列时, 则试图让每个组件的宽度相同。

程序清单 11.2 中的应用程序 Stacker 中包含一个面板, 面板中的按钮以方框布局的方式排列。

#### 程序清单 11.2 完整的 Stacker.java 源代码

```
1: import java.awt.*;
2: import javax.swing.*;
3:
4: public class Stacker extends JFrame {
5:     public Stacker() {
6:         super("Stacker");
7:         setSize(430, 150);
8:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9:         // create top panel
10:        JPanel commandPane = new JPanel();
11:        BoxLayout horizontal = new BoxLayout(commandPane,
12:            BoxLayout.X_AXIS);
13:        commandPane.setLayout(horizontal);
14:        JButton subscribe = new JButton("Subscribe");
15:        JButton unsubscribe = new JButton("Unsubscribe");
16:        JButton refresh = new JButton("Refresh");
17:        JButton save = new JButton("Save");
```

```

18:         commandPane.add(subscribe);
19:         commandPane.add(unsubscribe);
20:         commandPane.add(refresh);
21:         commandPane.add(save);
22:         // create bottom panel
23:         JPanel textPane = new JPanel();
24:         JTextArea text = new JTextArea(4, 70);
25:         JScrollPane scrollPane = new JScrollPane(text);
26:         // put them together
27:         FlowLayout flow = new FlowLayout();
28:         setLayout(flow);
29:         add(commandPane);
30:         add(scrollPane);
31:         setVisible(true);
32:     }
33:
34:     public static void main(String[] arguments) {
35:         Stacker st = new Stacker();
36:     }
37: }

```

编译并运行该程序时，结果如图 11.2 所示。

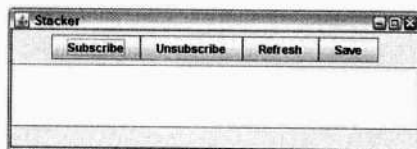


图 11.2 按钮由方框布局管理器管理的用户界面

按钮位于界面顶端，呈水平排列。如果在第 11~12 行中，传递给构造函数的第二个参数为 `BoxLayout.Y_AXIS`，按钮将垂直排列。

### 11.1.4 网格布局

网格布局管理器将组件放置到由行和列组成的网格中。首先，组件被加入到网格的第一行，并从最左边的单元格开始，依次向右排列。第一行的单元格排满后，接下来的组件将加入到第二行最左边的单元格中（如果有第二行的话），并以此类推。

网格布局是使用类 `GridLayout` 创建的，这个类位于 `java.awt` 包中。调用构造函数 `GridLayout` 时，提供两个参数：网格的行数和列数。下面的语句创建一个 10 行 3 列的网格布局管理器：

```
GridLayout gr = new GridLayout(10, 3);
```

与顺序布局一样，也可以提供另外两个参数，以指定组件间的垂直间距和水平间距。下面的语句创建一个 10 行 3 列的网格布局，并将水平间距和垂直间距分别设置为 5 像素和 8 像素：

```
GridLayout gr2 = new GridLayout(10, 3, 5, 8);
```

默认情况下，网格布局组件间的垂直间距和水平间距都为 0。

程序清单 11.3 中的应用程序创建了一个 3 行 3 列的网格，并将组件间的水平间距和垂直间距都设置为 10 个像素。

#### 程序清单 11.3 完整的 `Bunch.java` 源代码

```

1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class Bunch extends JFrame {
6:     JButton marcia = new JButton("Marcia");
7:     JButton carol = new JButton("Carol");

```



```

8:   JButton greg = new JButton("Greg");
9:   JButton jan = new JButton("Jan");
10:  JButton alice = new JButton("Alice");
11:  JButton peter = new JButton("Peter");
12:  JButton cindy = new JButton("Cindy");
13:  JButton mike = new JButton("Mike");
14:  JButton bobby = new JButton("Bobby");
15:
16:  public Bunch() {
17:      super("Bunch");
18:      setSize(260, 260);
19:      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20:      JPanel pane = new JPanel();
21:      GridLayout family = new GridLayout(3, 3, 10, 10);
22:      pane.setLayout(family);
23:      pane.add(marcia);
24:      pane.add(carol);
25:      pane.add(greg);
26:      pane.add(jan);
27:      pane.add(alice);
28:      pane.add(peter);
29:      pane.add(cindy);
30:      pane.add(mike);
31:      pane.add(bobby);
32:      add(pane);
33:      setVisible(true);
34:  }
35:
36:  public static void main(String[] arguments) {
37:      Bunch frame = new Bunch();
38:  }
39: }

```

图 11.3 显示了该应用程序的运行情况。



图 11.3 以 3×3 的网格布局方式排列 9 个按钮

对于图 11.3 中的按钮，需要指出的一点是，它们将扩展以填满相应的单元格。这是网格布局与其他布局管理器（显示的组件小得多）之间的一个重要差别。

### 11.1.5 边框布局

边框布局 (border layout) 是使用 java.awt 包中的 BorderLayout 类创建的，它将容器分成 5 部分：北、南、东、西和中央。图 11.4 说明了这 5 个部分是如何排列的。

在边框布局中，位于 4 个罗盘方位的组件将根据其需要占据相应的空间，余下的空间则属于中央区域。通常，这样的结果是，中央是一个大组件，四周是 4 个小组件。

要创建边框布局，可使用构造函数 BorderLayout() 或 BorderLayout(int, int)。在前者创建的边框布局中，组件间的间距为零，而后者指定了水平间距和垂直间距。

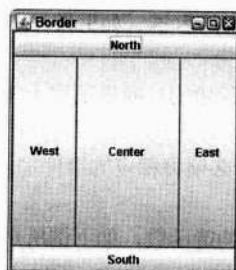


图 11.4 以边框布局方式排列的组件

创建边框布局并将其指定为某个容器的布局管理器后,便可以调用方法 `add()` 来将组件添加到该容器中,但该方法的调用方式与前面介绍的有所不同:

```
add(Component, String)
```

其中,第一个参数是要加入到容器中的组件。

第二个参数是 `BorderLayout` 的一个类变量,指定将组件加入到边框布局的哪个区域。可用的变量包括: `NORTH`、`SOUTH`、`EAST`、`WEST` 和 `CENTER`。

下面的语句将一个名为 `quitButton` 的按钮加入到边框布局的 `NORTH` 区域:

```
add(quitButton, BorderLayout.NORTH);
```

图 11.4 是由程序清单 11.4 中的应用程序生成的。

#### 程序清单 11.4 完整的 `Border.java` 源代码

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class Border extends JFrame {
6:     JButton nButton = new JButton("North");
7:     JButton sButton = new JButton("South");
8:     JButton eButton = new JButton("East");
9:     JButton wButton = new JButton("West");
10:    JButton cButton = new JButton("Center");
11:
12:    public Border() {
13:        super("Border");
14:        setSize(240, 280);
15:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16:        setLayout(new BorderLayout());
17:        add(nButton, BorderLayout.NORTH);
18:        add(sButton, BorderLayout.SOUTH);
19:        add(eButton, BorderLayout.EAST);
20:        add(wButton, BorderLayout.WEST);
21:        add(cButton, BorderLayout.CENTER);
22:    }
23:
24:    public static void main(String[] arguments) {
25:        Border frame = new Border();
26:        frame.setVisible(true);
27:    }
28: }
```

## 11.2 使用多个布局管理器

现在,您可能会问,Java 布局管理器对设计图形用户界面有何帮助呢?选择布局管理器是一个不断尝试的过程。

为找到合适的布局,常常需要对同一个界面组合使用多个布局管理器。

这是通过将多个容器加入到一个更大的容器（如框架）中，并给每个小容器指定布局管理器来实现的。

小容器为面板，它是使用 `JPanel` 类创建的。面板是用于将组件组合在一起的容器。使用面板时需要注意两点：

- 将面板加入到更大容器之前，必须将相应的组件加入到面板中；
- 面板有它自己的布局管理器。

要创建面板，可以调用 `JPanel` 类的构造函数，如下例所示：

```
JPanel pane = new JPanel();
```

要设置面板的布局，可调用其 `setLayout()` 方法。

下面的语句创建一个布局管理器，并将其用于名为 `pane` 的 `JPanel` 对象：

```
FlowLayout flo = new FlowLayout();
pane.setLayout(flo);
```

要将组件加入到面板中，可调用面板的 `add()` 方法，这与其他容器相同。

下面的语句创建一个文本框，并将其加入到名为 `pane` 的 `Panel` 对象中：

```
JTextField nameField = new JTextField(80);
pane.add(nameField);
```

本章后面的示例程序多次使用了面板。

### 11.3 卡片布局

卡片布局（card layout）不同于其他布局，因为它将隐藏一些组件。卡片布局是一组容器或组件，每次只显示其中的一个（这就像 21 点游戏，发牌人每次只翻开一张牌）。其中的容器被称为卡片。

如果您使用过 Macintosh 中的 HyperCard 软件或安装程序中的向导，则接触过使用卡片布局的程序。

使用卡片布局时，最常见的方式是使用面板作为卡片。首先将组件加入到面板中，然后将面板加入到使用卡片布局的容器中。

要创建卡片布局，可调用 `java.awt` 包中 `CardLayout` 类的构造函数：

```
CardLayout cc = new CardLayout();
```

要将这种布局管理器用于容器，可调用容器的 `setLayout` 方法，如下面的语句所示：

```
setLayout(cc);
```

将容器设置为使用卡片布局管理器后，必须使用 `add()` 方法将卡片加入容器中，该方法的调用方式与前面稍有不同。

使用的方法是 `add(Component, String)`。其中第一个参数指定将被用作卡片的容器或组件。如果是容器，则将其作为卡片加入之前，它必须包含所需的组件。

方法 `add()` 的第二个参数是一个字符串，表示卡片的名称。这可以是任何您想为卡片选取的名称。您可能想以某种方式对卡片进行编号，并在名称中使用这些编号，如“Card 1”、“Card 2”、“Card 3”等。

下面的语句将一个名为 `options` 的面板加入到容器中，并将该卡片命名为“Option Card”：

```
add(options, "Options Card");
```

使用卡片布局的容器首次被显示时，可见的是第一个被加入到容器中的卡片。

要显示其他的卡片，可调用卡片布局管理器的 `show()` 方法，该方法接受两个参数：

- 用于放置所有卡片的容器；
- 卡片的名称。

下面的语句调用名为 `cc` 的卡片布局管理器的 `show()` 方法：

```
cc.show(this, "Fact Card");
```

关键字 `this` 指的是包含该语句的对象，“Fact Card”是要显示的卡片的名称。显示卡片后，以前显

示的卡片将被隐藏。在卡片布局中，每次只能显示一张卡片。

在使用卡片布局管理器的程序中，卡片之间的切换通常是由用户的操作触发的。例如，对于在不同的卡片上显示邮送地址的程序，用户可以通过选择滚动列表中的选项，来显示不同的卡片。

## 在应用程序中使用卡片布局

接下来的项目演示卡片布局的用法以及如何在同一个图形用户界面中使用不同的布局管理器。

SurveyWizard 类是一个面板，它实现了向导界面：提出一系列简单问题，并通过按钮 Next 进入下一个问题。进入最后一个问题后，按钮为 Finish。

图 11.5 显示了这个面板。

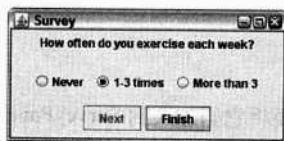


图 11.5 使用卡片布局实现类似于向导的界面

实现基于卡片的布局时，最简单的方法是使用面板。这个项目使用了多个面板：

- SurveyWizard 类是一个用于存储所有卡片的面板；
- 助手类 SurveyPanel 是一个存储一个卡片的面板；
- 每个 SurveyPanel 对象都包含 3 个面板，这些面板被堆积起来。

SurveyWizard 和 SurveyPanel 都是面板，因此采用卡片布局时，最简单的方法是使用面板。每个卡片都是一个面板，它们被加入到容器面板中，后者用于依次显示这些卡片。

上述工作是在构造函数 SurveyWizard 中完成的，它使用两个实例变量：一个是卡片布局管理器，另一个是一个包含三个 SurveyPanel 对象的数组：

```
SurveyPanel[] ask = new SurveyPanel[3];
CardLayout cards = new CardLayout();

构造函数设置类要使用的布局管理器，创建每个 SurveyPanel 对象，然后将它们加入到类中：
setLayout(cards);
String question1 = "What is your gender?";
String[] responses1 = { "female", "male", "not telling" };
ask[0] = new SurveyPanel(question1, responses1, 2);
add(ask[0], "Card 0");
```

创建每个 SurveyPanel 对象时，都给构造函数提供了 3 个参数：问题的内容、答案数组和默认答案的编号。

在上述代码中，问题为 “What is your gender?”，可能的答案为 “female”、“male” 或 “not telling”。默认答案为第 3 个 “not telling”。

构造函数 SurveyPanel 使用一个标签组件来存储问题和一组单选按钮来存储可能的答案：

```
SurveyPanel(String ques, String[] resp, int def) {
    question = new JLabel(ques);
    response = new JRadioButton[resp.length];
    // more to come
}
```

这个类使用网格布局来将组件排列一行三列。加入到每个网格中的组件都是面板。

首先，创建了一个用于容纳问题标签的面板：

```
JPanel sub1 = new JPanel();
JLabel quesLabel = new JLabel(ques);
sub1.add(quesLabel);
```

标签在面板中的位置是由面板的默认布局（顺序布局和居中）决定的。

接下来创建一个用于容纳可选答案的面板。使用一个 for 循环来遍历存储了可选答案的字符串数组。这些数组元素被用于创建单选按钮。构造函数 JRadioButton 的第二个参数指定单选按钮是否被选中。完成这些工作的代码如下：

```
JPanel sub2 = new JPanel();
for (int i = 0; i < resp.length; i++) {
    if (def == i) {
        response[i] = new JRadioButton(resp[i], true);
    } else {
        response[i] = new JRadioButton(resp[i], false);
    }
    group.add(response[i]);
    sub2.add(response[i]);
}
```

最后一个面板用于存放按钮 Next 和 Finish：

```
JPanel sub3 = new JPanel();
nextButton.setEnabled(true);
sub3.add(nextButton);
finalButton.setEnabled(false);
sub3.add(finalButton);
```

设置好了所需的 3 个面板后，需要将它们加入到 SurveyPanel 中，这是构造函数需要完成的最后一项工作：

```
GridLayout grid = new GridLayout(3, 1);
setLayout(grid);
add(sub1);
add(sub2);
add(sub3);
```

SurveyPanel 类中还有一项内容：一个这样的方法，它在到达最后一个问题时启用 Finish 按钮并禁用 Next 按钮：

```
void setFinalQuestion(boolean finalQuestion) {
    if (finalQuestion) {
        nextButton.setEnabled(false);
        finalButton.setEnabled(true);
    }
}
```

在使用卡片布局的用户界面中，每个卡片的显示都是用户执行操作的结果。

这些操作被称为事件，这将在第 12 章中介绍。

这里简要地介绍一下 SurveyPanel 类如何对用户单击按钮做出响应。

这个类实现了 ActionListener——java.awt.event 包中的一个接口：

```
public class SurveyWizard extends JPanel implements ActionListener {
    // more to come
}
```

这个接口让这个类能够对事件（单击按钮、选择菜单和类似的用户输入）做出响应。

接下来调用每个按钮的 addActionListener(Object)方法：

```
ask[0].nextButton.addActionListener(this);
ask[0].finalButton.addActionListener(this);
```

监听器 (Listener) 是监视特定用户输入的类型。传递给 addActionListener(Object)方法的参数是关注事件的类，将参数设置为 this 表明这种工作将由 SurveyPanel 处理。

接口 ActionListener 只有一个方法：

```
public void actionPerformed(ActionEvent evt) {
    // more to come
}
```

被监听的组件发生事件后，这个方法将被调用。在 SurveyPanel 中，每当按钮被单击时都将调用这个方法。

在 SurveyPanel 中，这个方法使用一个实例变量来指出应显示哪个卡片：

```
int currentCard = 0;
```

每当按钮被单击，导致 actionPerformed()被调用时，这个变量的都加 1。然后，调用卡片管理的

show(Container, String)方法来显示新的卡片。显示最后一张卡片后, Next 按钮将被禁用。

下面是这个方法的完整源代码:

```
public void actionPerformed(ActionEvent evt) {
    currentCard++;
    if (currentCard >= ask.length) {
        ask[2].finalButton.setEnabled(false);
    }
    cards.show(this, "Card " + currentCard);
}
```

程序清单 11.5 是 SurveyWizard 类的完整源代码。

#### 程序清单 11.5 完整的 SurveyWizard.java 源代码

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class SurveyWizard extends JPanel implements ActionListener {
6:     int currentCard = 0;
7:     CardLayout cards = new CardLayout();
8:     SurveyPanel[] ask = new SurveyPanel[3];
9:
10:    public SurveyWizard() {
11:        super();
12:        setSize(240, 140);
13:        setLayout(cards);
14:        // set up survey
15:        String question1 = "What is your gender?";
16:        String[] responses1 = { "female", "male", "not telling" };
17:        ask[0] = new SurveyPanel(question1, responses1, 2);
18:        String question2 = "What is your age?";
19:        String[] responses2 = { "Under 25", "25-34", "35-54",
20:                                "Over 54" };
21:        ask[1] = new SurveyPanel(question2, responses2, 1);
22:        String question3 = "How often do you exercise each week?";
23:        String[] responses3 = { "Never", "1-3 times", "More than 3" };
24:        ask[2] = new SurveyPanel(question3, responses3, 1);
25:        ask[2].setFinalQuestion(true);
26:        for (int i = 0; i < ask.length; i++) {
27:            ask[i].nextButton.addActionListener(this);
28:            ask[i].finalButton.addActionListener(this);
29:            add(ask[i], "Card " + i);
30:        }
31:    }
32:
33:    public void actionPerformed(ActionEvent evt) {
34:        currentCard++;
35:        if (currentCard >= ask.length) {
36:            System.exit(0);
37:        }
38:        cards.show(this, "Card " + currentCard);
39:    }
40: }
41:
42: class SurveyPanel extends JPanel {
43:     JLabel question;
44:     JRadioButton[] response;
45:     JButton nextButton = new JButton("Next");
46:     JButton finalButton = new JButton("Finish");
47:
48:     SurveyPanel(String ques, String[] resp, int def) {
49:         super();
50:         setSize(160, 110);
51:         question = new JLabel(ques);
52:         response = new JRadioButton[resp.length];
53:         JPanel sub1 = new JPanel();
54:         ButtonGroup group = new ButtonGroup();
55:         JLabel quesLabel = new JLabel(ques);
56:         sub1.add(quesLabel);
57:         JPanel sub2 = new JPanel();
```

```

58:         for (int i = 0; i < resp.length; i++) {
59:             if (def == i) {
60:                 response[i] = new JRadioButton(resp[i], true);
61:             } else {
62:                 response[i] = new JRadioButton(resp[i], false);
63:             }
64:             group.add(response[i]);
65:             sub2.add(response[i]);
66:         }
67:         JPanel sub3 = new JPanel();
68:         nextButton.setEnabled(true);
69:         sub3.add(nextButton);
70:         finalButton.setEnabled(false);
71:         sub3.add(finalButton);
72:         GridLayout grid = new GridLayout(3, 1);
73:         setLayout(grid);
74:         add(sub1);
75:         add(sub2);
76:         add(sub3);
77:     }
78:
79:     void setFinalQuestion(boolean finalQuestion) {
80:         if (finalQuestion) {
81:             nextButton.setEnabled(false);
82:             finalButton.setEnabled(true);
83:         }
84:     }
85: }

```

编译 SurveyWizard 类后，便可以将其加入到任何 Swing 用户界面中。

程序清单 11.6 是一个简单的框架应用程序，它显示调查（survey）面板。

程序清单 11.6 完整的 SurveyFrame.java 源代码

```

1: import java.awt.*;
2: import javax.swing.*;
3:
4: public class SurveyFrame extends JFrame {
5:     public SurveyFrame() {
6:         super("Survey");
7:         setSize(290, 140);
8:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9:         SurveyWizard wiz = new SurveyWizard();
10:        add(wiz);
11:        setVisible(true);
12:    }
13:
14:    public static void main(String[] arguments) {
15:        SurveyFrame surv = new SurveyFrame();
16:    }
17: }

```

这个应用程序的运行情况如图 11.5 所示。

## 11.4 网格袋布局

Java 中的最后一种布局管理器是网格袋（grid bag layout）布局，它扩展了网格布局管理器。网格袋布局与网格布局之间的区别在于：

- 组件可占据网格中的多个单元格；
- 不同行和列的比例不必相等；
- 组件不必填满单元格；
- 单元格中的组件可以以不同的方式对齐。

要采用网格袋布局，需要使用 GridBagLayout 和 GridBagConstraints 类，它们都位于 java.awt 包中。

GridBagLayout 是布局管理器，而 GridBagConstraints 定义如何在网格中放置组件。

网格袋布局管理器的构造函数不接受任何参数，和其他管理器一样，该构造函数也可用于容器。

在框架或窗口的构造函数中，可使用下面的语句来指定使用网格袋布局：

```
Container pane = getContentPane();
GridBagLayout bag = new GridBagLayout();
pane.setLayout(bag);
```

在网格袋布局中，每个组件都使用一个 GridBagConstraints 对象来指定组件将占用网格的哪些单元格、组件的大小以及组件的其他显示属性。

GridBagConstraints 对象有 11 个实例变量，它们决定了放置组件的方式。

- gridx: 容纳组件的单元格的 x 位置，如果组件跨越了多个单元格，则为组件左上角所处的单元格的 x 位置。
- gridy: 单元格或左上角单元格的 y 位置。
- gridwidth: 组件沿水平方向跨越了多少个单元格。
- gridheight: 组件沿垂直方向跨越了多少个单元格。
- weightx: 组件相对于同一行中其他组件的大小。
- weighty: 组件相对于同一列中其他组件的大小。
- anchor: 指定组件显示在单元格内的什么位置（如果组件没有填满整个单元格）。
- fill: 指定是沿水平还是垂直方向扩大组件，使之填满单元格。
- insets: 一个 Insets 对象，设置组件和单元格边界之间的空白。
- ipadx: 组件宽度的放大量。
- ipady: 组件高度的放大量。

除 insets 外，其他实例变量都是整数值。要使用这个类，最简单的方法是不使用任何参数创建一个约束条件对象，然后分别设置其每个变量的值。没有被显式地设置时，变量将使用默认值。

下面的代码创建一个网格袋布局对象和一个约束条件对象，用于在网格中放置组件：

```
Container pane = getContentPane();
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
pane.setLayout(gridbag);
```

可以使用一组赋值语句来设置约束条件对象：

```
constraints.gridx = 0;
constraints.gridy = 0;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.weightx = 100;
constraints.weighty = 100;
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
```

上述代码设置了这样的约束条件：将组件放在网格位置(0, 0)处，其宽为 2 个单元格，高为一个单元格。

使用 GridBagConstraints 的类变量设置了组件的大小和位置，组件将居中(anchor 的值为 CENTER)，且不会扩大以填满单元格(fill 的值为 NONE)。

weightx 和 weighty 只有同其他组件的相应值进行比较时才有意义，本节后面将详细介绍它们。

将组件加入网格袋布局中分两步：

1. 调用布局管理器的 setConstraints(Component, GridBagConstraints)方法，并将组件和约束条件对象作为参数；
2. 将组件加入到使用该管理器的容器中。

下面的语句继续前面的例子，将一个按钮加入到面板中：

```
JButton okButton = new JButton("OK");
gridbag.setConstraints(okButton, constraints);
pane.add(okButton);
```



在网格中放置每个组件之前，必须设置约束条件对象。

### 11.4.1 设计网格

由于网格布局很复杂，因此使用它之前做些准备工作将会有所帮助；为此，可以在纸上画出草图或者以其他方式进行说明。

图 11.6 是一个 E-mail 程序的用户界面中的面板草图。

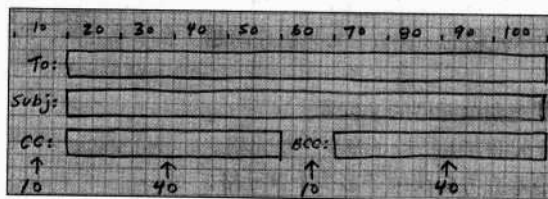


图 11.6 在网格上设计用户界面

图 11.6 中的面板中包含一组标签和文本框，发送邮件时用户需要填写这些文本框。

这种界面适合使用网格袋布局，因为其中的组件的宽度各不相同。所有标签的宽度都相同，但文本框 To 和 Subject 比 CC 和 BCC 宽。在网格袋布局中，每个组件单独占用不同的单元格，不能与其他组件共用单元格。组件可能占用多个单元格。

图 11.6 中的草图没有指出单元格，但使用 0~100 的值指出了组件的宽度。这些值是相对值，而不是组件的绝对大小，它们可用于计算 `weightx` 和 `weighty` 的值。

#### 注意

此时您可能会问，草图中为何没有在垂直方向上提供 0~100 的比值。E-mail 程序界面不需要这样的值——所有组件的高度都相同（因此 `weighty` 也相同）。

绘制出用户界面的草图，指出组件的相对大小后，便可以确定每个组件的单元格位置和大小。

在这个 Email 程序界面中，每个组件的宽度都被设置为 10 的倍数，因此可以使用一个包括 10 列的网格。

和网格布局一样，左上角的单元格为(0,0)。x 坐标表示列，而 y 坐标表示行。向右下移动时，它们的值将增加。

图 11.7 指出了每个组件的位置(x, y)和宽度，单位为单元格。

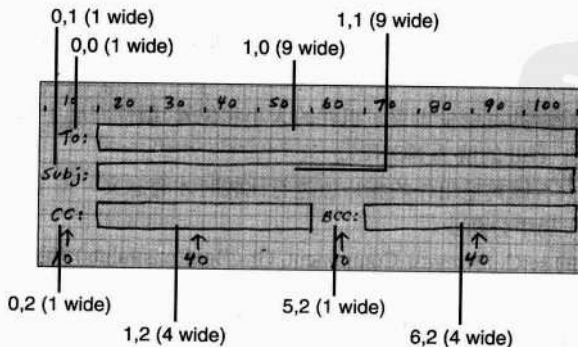


图 11.7 组件在网格中占据的单元格

## 11.4.2 创建网格

在图纸上绘制好草图后，便可以编写实现用户界面所需的代码。

下述位于 E-mail 面板的构造函数中的语句将面板设置为使用网格袋布局，并将标签 To 和对应的文本框加入到面板中：

```
public MessagePanel() {
    GridBagLayout gridbag = new GridBagLayout();
    setLayout(gridbag);
    // add the label
    JLabel toLabel = new JLabel("To: ");
    GridBagConstraints constraints = new GridBagConstraints();
    constraints.gridx = 0;
    constraints.gridy = 0;
    constraints.gridwidth = 1;
    constraints.gridheight = 1;
    constraints.weightx = 10;
    constraints.weighty = 100;
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.EAST;
    gridbag.setConstraints(toLabel, constraints);
    add(toLabel);
    // add the text field
    JTextField to = new JTextField();
    constraints = new GridBagConstraints();
    constraints.gridx = 1;
    constraints.gridy = 0;
    constraints.gridwidth = 9;
    constraints.gridheight = 1;
    constraints.weightx = 90;
    constraints.weighty = 100;
    constraints.fill = GridBagConstraints.HORIZONTAL;
    constraints.anchor = GridBagConstraints.WEST;
    gridbag.setConstraints(to, constraints);
    add(to);
}
```

标签和文本框使用各自的约束条件对象（重用变量 constraints）。gridx 和 gridy 的值使得标签和文本框分别被放置在(0, 0)和(1,0)处。gridwidth 的值使得标签和文本框的宽度分别是 1 和 9 个单元格。

它们使用的 fill 值不同：标签使用的是 NONE，因此它不会扩大，而文本框使用的是 HORIZONTAL，因此它只会沿水平方向扩大（另外两个可用的值是 VERTICAL 和 BOTH）。

它们使用的 anchor 值也不同。标签使用的是类变量 EAST，因此与单元格的右边界对齐；而文本框使用的是 WEST，因此与左边界对齐。

可以使用所有的罗盘方向以及 CENTER，这包括 NORTH、NORTHEAST、EAST、SOUTHEAST、SOUTH、SOUTHWEST、WEST 和 NORTHWEST。

在网格袋约束条件中，最复杂的是 weightx 和 weighty。这些变量可以是任何的整数或 double 值，它们指出了组件之间的相对大小。

标签 To 的 weightx 值为 10，而与之相邻的文本框的 weightx 值为 90，比例与图 11.5 所示的草图相同。这些值使得文本框的宽度为标签的 9 倍。这些值是任意的，如果标签的 weightx 为 3，而文本框为 27，则文本框的宽度仍为标签的 9 倍。

如果不需要给组件提供不同的权重，应对同一行（列）中的组件使用相同的值。例如，同一列中的标签和文本框的 weighty 都为 100，因此它们的高度相同。

设置网格袋约束条件需要使用大量重复的代码。为减少输入量，在 E-mail 面板类中提供了一个方法，用于设置组件的约束条件，并将组件加入到面板中：

```
private void addComponent(Component component, int gridx, int gridy,
    int gridwidth, int gridheight, int weightx, int weighty, int fill,
    int anchor) {
    GridBagConstraints constraints = new GridBagConstraints();
```

```

constraints.gridx = gridx;
constraints.gridy = gridy;
constraints.gridwidth = gridwidth;
constraints.gridheight = gridheight;
constraints.weightx = weightx;
constraints.weighty = weighty;
constraints.fill = fill;
constraints.anchor = anchor;
gridbag.setConstraints(component, constraints);
add(component);
}

```

这个方法可用于所有这样的面板，即它使用存储在实例变量 `gridbag` 中的 `GridBag Layout` 管理器。它没有使用 `GridBagConstraints` 类中的变量 `insets`、`ipadx` 和 `ipady`，因此这些变量将保留默认值。

下面的语句调用这个方法将标签 `Subject` 及其对应的文本框加入到面板中：

```

JLabel subjectLabel = new JLabel("Subject: ");
addComponent(subjectLabel, 0, 1, 1, 1, 10, 100, GridBagConstraints.NONE,
    GridBagConstraints.EAST);
JTextField subject = new JTextField();
addComponent(subject, 1, 1, 9, 1, 90, 100, GridBagConstraints.HORIZONTAL,
    GridBagConstraints.WEST);

```

最后，下面的语句将标签 `CC` 和 `BCC` 及其对应的文本框加入到面板中：

```

// add a CC label at (0,2) 1 cell wide
JLabel ccLabel = new JLabel("CC: ");
addComponent(ccLabel, 0, 2, 1, 1, 10, 100, GridBagConstraints.NONE,
    GridBagConstraints.EAST);
// add a CC text field at (1,2) 4 cells wide
JTextField cc = new JTextField();
addComponent(cc, 1, 2, 4, 1, 40, 100, GridBagConstraints.HORIZONTAL,
    GridBagConstraints.WEST);
// add a BCC label at (5,2) 4 cells wide
JLabel bccLabel = new JLabel("BCC: ");
addComponent(bccLabel, 5, 2, 1, 1, 10, 100, GridBagConstraints.NONE,
    GridBagConstraints.EAST);
// add a BCC text field at (6,2) 4 cells wide
JTextField bcc = new JTextField();
addComponent(bcc, 6, 2, 4, 1, 40, 100, GridBagConstraints.HORIZONTAL,
    GridBagConstraints.WEST);

```

这 4 个组件位于同一行中，这使得它们的 `weightx` 值至关重要。标签的 `weightx` 值被设置为 10，而文本框的 `weightx` 值被设置为 40，这与草图中一致。

程序清单 11.7 是 E-mail 面板类 (`MessagePanel`) 的完整源代码。

#### 程序清单 11.7 完整的 `MessagePanel.java` 源代码

```

1: import java.awt.*;
2: import javax.swing.*;
3:
4: public class MessagePanel extends JPanel {
5:     GridBagLayout gridbag = new GridBagLayout();
6:
7:     public MessagePanel() {
8:         super();
9:         GridBagConstraints constraints;
10:        setLayout(gridbag);
11:
12:        JLabel toLabel = new JLabel("To: ");
13:        JTextField to = new JTextField();
14:        JLabel subjectLabel = new JLabel("Subject: ");
15:        JTextField subject = new JTextField();
16:        JLabel ccLabel = new JLabel("CC: ");
17:        JTextField cc = new JTextField();
18:        JLabel bccLabel = new JLabel("BCC: ");
19:        JTextField bcc = new JTextField();
20:
21:        addComponent(toLabel, 0, 0, 1, 1, 10, 100,
22:            GridBagConstraints.NONE, GridBagConstraints.EAST);
23:        addComponent(to, 1, 0, 9, 1, 90, 100,

```

```

24:         GridBagConstraints.HORIZONTAL, GridBagConstraints.WEST);
25:     addComponent(subjectLabel, 0, 1, 1, 1, 10, 100,
26:         GridBagConstraints.NONE, GridBagConstraints.EAST);
27:     addComponent(subject, 1, 1, 9, 1, 90, 100,
28:         GridBagConstraints.HORIZONTAL, GridBagConstraints.WEST);
29:     addComponent(ccLabel, 0, 2, 1, 1, 10, 100,
30:         GridBagConstraints.NONE, GridBagConstraints.EAST);
31:     addComponent(cc, 1, 2, 4, 1, 40, 100,
32:         GridBagConstraints.HORIZONTAL, GridBagConstraints.WEST);
33:     addComponent(bccLabel, 5, 2, 1, 1, 10, 100,
34:         GridBagConstraints.NONE, GridBagConstraints.EAST);
35:     addComponent(bcc, 6, 2, 4, 1, 40, 100,
36:         GridBagConstraints.HORIZONTAL, GridBagConstraints.WEST);
37: }
38:
39: private void addComponent(Component component, int gridx, int gridy,
40:     int gridwidth, int gridheight, int weightx, int weighty, int fill,
41:     int anchor) {
42:
43:     GridBagConstraints constraints = new GridBagConstraints();
44:     constraints.gridx = gridx;
45:     constraints.gridy = gridy;
46:     constraints.gridwidth = gridwidth;
47:     constraints.gridheight = gridheight;
48:     constraints.weightx = weightx;
49:     constraints.weighty = weighty;
50:     constraints.fill = fill;
51:     constraints.anchor = anchor;
52:     gridbag.setConstraints(component, constraints);
53:     add(component);
54: }
55: }

```

编译这个类后，便可以将其用于任何用户界面中（估计它会被加入到 E-mail 程序中用于写信的界面中）。

图 11.8 是将其加入到一个宽 320 像素、高 120 像素的框架中时的外观。

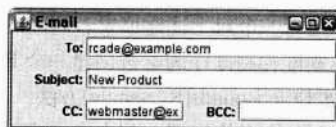


图 11.8 应用程序用户界面中的上述面板

由于面板不能指定自己的大小，因此其大小取决于框架的大小。这种灵活性展示了 Swing 的网格布局和网格袋布局的强大之处：让组件能够适应界面中可用的空间。

### 11.4.3 单元格 padding 和 insets

在上述 E-mail 面板示例中，没有使用 GridBagConstraints 中的 3 个变量：insets、ipadx 和 ipady。约束条件 ipadx 和 ipady 控制 padding（在组件周围的空白）。默认情况下，组件的周围都没有空白（通过填满单元格的组件最容易了解这一点）。ipadx 在组件的两边加入空白，ipady 在的组件上下加入空白。

创建新的布局管理器时，组件水平和垂直间距（在网格袋布局中使用 ipadx 和 ipady）决定面板内组件间的间隔大小。Insets 决定面板周围的空白大小。类 insets 包含上、下、左和右 inset 值，绘制面板本身时，将使用这些值。

insets 决定面板边界与面板内组件之间的间隔。

下面的语句创建一个 Insets 对象，指定上和下为 20 像素，左和右为 13 像素。

```
Insets whitespace = new Insets(20, 13, 20, 13);
```

在容器中，可以通过覆盖 `getInsets()` 方法，并返回一个 `Insets` 对象来指定 insets，如下例所示：

```
public Insets getInsets() {
    return new Insets(10, 30, 10, 30);
}
```

## 11.5 总结

正如您在本章中看到的，抽象表现主义的作用有限。对于习惯于更精确地控制组件在界面中的位置的人而言，要使用布局管理器，必须调整其观念。

您现在知道如何使用 5 种不同的布局管理器和面板。当您使用 AWT 时将发现，通过使用嵌套容器和不同的布局管理器，几乎可以实现任何类型的界面。

掌握如何使用 Java 来开发用户界面后，您的程序便可以提供大多数其他可视化编程语言不能提供的东西：无需修改就能够运行在多个平台上的界面。

## 11.6 问与答

问：我不喜欢使用布局管理器，它们要么太简单，要么太复杂（如网格袋布局）。即使经过大量的调整，用户界面也达不到我的要求。我想要做的是，定义组件大小并将它们放到屏幕的 (x,y) 位置上。我能做到这一点吗？

答：这是可能的，但会有很多问题。Java 被设计成使程序的图形用户界面能够在不同系统上运行，这些系统的平台、分辨率、字体、屏幕尺寸等可以不同。依赖于像素坐标可能导致这样的问题，即在某个平台可能很好，但却不能用于其他平台上，在这些平台上，组件可能彼此重叠、被容器边缘截掉一部分等。布局管理器动态地在屏幕上放置组件，从而避免了这些问题。虽然在不同的平台上，最终的结果可能会有所不同，但这种不同不会是灾难性的。

如果上述问题还不足以说服您，来看一个对上述建议置若罔闻的后果：用 `null` 作为参数来设置内容窗格的布局管理器，以 `x`、`y` 坐标以及宽度和高度作为参数创建一个 `Rectangle`（位于 `java.awt` 包中）对象，然后以该 `Rectangle` 作为参数调用组件的 `setBounds(Rectangle)` 方法。

下面的应用程序显示一个  $300 \times 300$  像素的框架，其中有一个“Click Me”按钮，位置为 (10, 10)，宽度和高度分别为 120 和 30 像素：

```
import java.awt.*;
import javax.swing.*;

public class Absolute extends JFrame {
    public Absolute() {
        super("Example");
        setSize(300, 300);
        Container pane = getContentPane();
        pane.setLayout(null);
        JButton myButton = new JButton("Click Me");
        myButton.setBounds(new Rectangle(10, 10, 120, 30));
        pane.add(myButton);
        getContentPane().add(myButton);
        setVisible(true);
    }

    public static void main(String[] arguments) {
        Absolute ex = new Absolute();
    }
}
```

有关 `setBounds()` 的更详细的信息，请参阅 `Component` 类。

## 11.7 小测验

请回答下述问题，以复习本章介绍的内容。

### 11.7.1 问题

1. 在 Java 中，面板的默认布局管理器是什么？
  - a. 没有；
  - b. BorderLayout；
  - c. FlowLayout。
2. 在将组件加入到容器中时，哪个布局管理器需要罗盘方向或中央位置？
  - a. BorderLayout；
  - b. MapLayout；
  - c. FlowLayout。
3. 如果希望网格布局中的组件可以占据多个单元格，应使用哪种布局？
  - a. GridLayout；
  - b. GridBagLayout；
  - c. 两者都不是，这种目标无法实现。

答案

1. c。
2. a。
3. b。

### 11.7.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

对于下述代码：

```
import java.awt.*;
import javax.swing.*;

public class ThreeButtons extends JFrame {
    public ThreeButtons() {
        super("Program");
        setSize(350, 225);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton alpha = new JButton("Alpha");
        JButton beta = new JButton("Beta");
        JButton gamma = new JButton("Gamma");
        JPanel content = new JPanel();
        // answer goes here
        content.add(alpha);
        content.add(beta);
        content.add(gamma);
        add(content);
        pack();
        setVisible(true);
    }

    public static void main(String[] arguments) {
        ThreeButtons b3 = new ThreeButtons();
    }
}
```



为使框架并排地显示所有的 3 个按钮，应将// answer goes here 替换为下述哪条语句？

- a. `content.setLayout(null)`
- b. `content.setLayout(new FlowLayout())`
- c. `content.setLayout(new GridLayout(3,1))`
- d. `content.setLayout(new BorderLayout())`

答案 b

## 11.8 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 创建一个用户界面，它显示了一个月的日历，最上面是表示星期和月份的标题。
2. 创建一个使用了多种布局管理器的界面。



## 第 12 章

# 响应用户输入

要使 Java 程序的界面充分发挥其作用，必须使之能够接收用户事件。

Swing 用一组被称为事件监听器（event listener）的接口来处理事件。您可以创建监听器对象，并将其与要监听的用户界面组件关联起来。

本章介绍如何将各种监听器加入到 Swing 程序，包括处理行为事件（action event）、鼠标事件和其他交互的监听器。

然后，使用 Swing 类创建一个完整的 Java 程序。

### 12.1 事件监听器

java.awt.event 包提供了所有基本的事件监听器以及表示特定事件的对象。下述监听器接口最有用：

- ActionListener：行为事件，由用户对组件执行某种操作（如单击按钮）激发。
- AdjustmentListener：调整事件，组件被调整（如移动滚动条上的滑块）时激发。
- FocusListener：键盘焦点事件，诸如文本框等组件获得或失去焦点时激发。
- ItemListener：选项事件，诸如复选框等选项被修改时激发。
- KeyListener：键盘事件，用户通过键盘输入文本时激发。
- MouseListener：鼠标事件，鼠标单击、鼠标移入或离开组件时激发。
- MouseMotionListener：鼠标移动事件，跟踪鼠标在组件上的运动。
- WindowListener：窗口事件，窗口（如应用程序的主窗口）被最大化、最小化、移动或关闭时激发。

类可以根据需要实现任意数目的监听器。下面的类被声明成能够处理行为事件和文本事件：

```
public class Suspense extends JFrame implements ActionListener,
    TextListener {
    // ...
}
```

要在程序中使用这些类，可以分别导入它们或使用下面的语句导入整个包：

```
import java.awt.event.*;
```

#### 12.1.1 设置组件

将类用作事件监听器时，必须首先设置该类要监听的事件类型。如果不进行第二步操作——将匹配的监听器加入到组件中，这种情况将不会发生。组件被使用时，该监听器将激发相应的事件。

创建组件后，可以调用组件的下述方法之一将监听器与组件关联起来。

- addActionListener()：JButton、Jcheck、JcomboBox、JTextField 和 JRadioButton 组件。
- addFocusListener()：所有 Swing 组件。



- `addItemListener()`: `JButton`、`JCheckBox`、`JComboBox` 和 `JRadioButton` 组件。
- `addKeyListener()`: 所有 Swing 组件。
- `addMouseListener()`: 所有 Swing 组件。
- `addMouseMotionListener()`: 所有 Swing 组件。
- `addTextListener()`: `JTextField` 和 `JTextArea` 组件。
- `addWindowListener()`: 所有 `JWindow` 和 `JFrame` 组件。

**警告**

在 Java 程序中，人们常犯的一个错误是，将组件加入到容器之后对其进行修改。将组件加入到容器之前，必须将监听器与组件关联起来，并完成其他配置工作。否则，当程序运行时，这些设置将被忽略。

下面的例子创建一个 `JButton` 对象，并将一个行为事件监听器与之关联起来：

```
JButton zap = new JButton("Zap");
zap.addActionListener(this);
```

所有这些 `add` 方法都接受一个参数：对事件进行监听的对象。`this` 表示当前类就是事件监听器。您可以指定其他对象，只要它的类实现了相应的监听器接口。

### 12.1.2 事件处理方法

将接口与类关联起来时，这个类必须处理接口中包含的所有方法。

就事件监听器而言，每个方法都是由窗口系统自动调用的，这是在对应的用户事件发生时进行的。

接口 `ActionListener` 只有一个方法：`actionPerformed()`。所有实现了 `ActionListener` 的类都必须有一个结构与下面类似的方法：

```
public void actionPerformed(ActionEvent event) {
    // handle event here
}
```

如果程序的图形用户界面中只有一个组件有行为事件的监听器，则 `actionPerformed()` 方法可用于响应由该组件激发的事件。

如果有多个组件有行为事件监听器，则必须根据 `ActionEvent` 对象来判断程序中哪个组件被使用，从而采取相应的措施。

您可能已经注意到了，方法 `actionPerformed()` 接受一个 `ActionEvent` 对象作为参数。该对象可以用于获悉激发事件的组件的细节。

`ActionEvent` 和所有其他事件对象都位于 `java.awt.event` 包中，它们都是 `EventObject` 的子类。

每种事件处理方法都接受某种事件对象作为参数。这种对象的方法 `getSource()` 可用来判断激发事件的组件，如下例所示：

```
public void actionPerformed(ActionEvent event) {
    Object source = evt.getSource();
}
```

可以使用运算符 `==` 将方法 `getSource()` 返回的对象与组件进行比较。下面的语句可用于前面的 `actionPerformed()` 方法中，以处理用户对按钮 `quitButton` 和 `sortRecords` 的单击：

```
if (source == quitButton) {
    quitProgram();
}
if (source == sortRecords) {
    sortRecords();
}
```

如果事件是由对象 `quitButton` 激发的，则调用方法 `quitProgram()`；如果是由对象 `sortRecord` 激发的，则调用方法 `sortRecords()`。

很多事件处理方法都针对每种事件或组件调用不同的方法。这使得事件处理方法更容易阅读。此

外，如果类中包含多个事件处理方法，则其中的每一个都可以调用相同的方法来完成工作。

在事件处理方法中，可以使用运算符 `instanceof` 来检查激发事件的组件类型。下面的代码可用于包含一个按钮和一个文本框的程序中，这两种组件都激发行事件：

```
void actionPerformed(ActionEvent event) {
    Object source = event.getSource();
    if (source instanceof JTextField) {
        calculateScore();
    } else if (source instanceof JButton) {
        quitProgram();
    }
}
```

程序清单 12.1 中的程序包含两个 `JButton` 组件框架，这些按钮用于修改框架的标题栏中的文本。

#### 程序清单 12.1 完整的 `TitleChanger.java` 源代码

```
1: import java.awt.event.*;
2: import javax.swing.*;
3: import java.awt.*;
4:
5: public class TitleChanger extends JFrame implements ActionListener {
6:     JButton b1 = new JButton("Rosencrantz");
7:     JButton b2 = new JButton("Guildenstern");
8:
9:     public TitleChanger() {
10:         super("Title Bar");
11:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:         b1.addActionListener(this);
13:         b2.addActionListener(this);
14:         FlowLayout flow = new FlowLayout();
15:         setLayout(flow);
16:         add(b1);
17:         add(b2);
18:         pack();
19:         setVisible(true);
20:     }
21:
22:
23:     public void actionPerformed(ActionEvent evt) {
24:         Object source = evt.getSource();
25:         if (source == b1) {
26:             setTitle("Rosencrantz");
27:         } else if (source == b2) {
28:             setTitle("Guildenstern");
29:         }
30:         repaint();
31:     }
32:
33:     public static void main(String[] arguments) {
34:         TitleChanger frame = new TitleChanger();
35:     }
36: }
```

使用 Java 解释器运行该应用程序时，其界面类似于图 12.1。

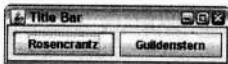


图 12.1 应用程序 `TitleChanger`

为响应该应用程序中的行为事件，只需要 12 行。

- 第 1 行导入 `java.awt.event` 包。
- 第 5 行实现接口 `ActionListener`。
- 第 12~13 行将行为监听器加入到两个按钮中。

- 第23~31行响应两个 JButton 对象上发生的行为事件。evt 对象的 getSource() 方法用于确定事件的来源。如果是按钮 b1, 则将框架的标题设置为 Rosencrantz; 如果为 b2, 则设置为 Guildenstern。repaint() 调用是必不可少的, 这样, 在方法中修改标题后, 将重绘框架。

## 12.2 使用方法

接下来的几节将详细介绍每个事件处理方法的构造以及可以在其中使用的方法。

除前面介绍的方法外, 还可以调用任何事件对象的 getSource() 方法来判断事件是由哪个对象激发的。

### 12.2.1 行为事件

行为事件在用户使用下面组件之一完成某种操作时发生: 按钮、复选框、菜单项、文本框和单选按钮。

要处理这些事件, 类必须实现接口 ActionListener。此外, 还必须对于每个要激发行为事件的组件调用方法 addActionListener(), 除非要忽略该组件的行为事件。

方法 actionPerformed(ActionEvent) 是接口 ActionListener 中唯一的一个方法, 其格式如下:

```
public void actionPerformed(ActionEvent event) {
    // ...
}
```

除方法 getSource() 外, 还可以调用 ActionEvent 对象的方法 getActionCommand() 来获得有关事件来源的更详细的信息。

默认情况下, 动作命令 (action command) 是与组件相关联的文本, 如按钮上的标签。您还可以调用组件的 setActionCommand(String) 方法来设置不同的动作命令。其中字符串参数是希望的动作命令文本。

例如, 下面的语句创建了一个按钮和一个菜单项, 并将它们的动作命令都指定为 "Sort Files":

```
JButton sort = new JButton("Sort");
JMenuItem menuSort = new JMenuItem("Sort");
sort.setActionCommand("Sort Files");
menuSort.setActionCommand("Sort Files");
```

#### 注意

当程序中的多个组件可能激发相同的事件时, 动作命令将很有用。例如, 程序中有一个 Quit 按钮, 同时其下拉菜单中有一个 Quit 选项。通过将相同的动作命令指定给这两个组件, 可以在事件处理方法中使用相同的代码来处理它们。

### 12.2.2 焦点事件

焦点事件在图形用户界面上的组件在获得或失去输入焦点时发生。获得焦点表明组件可以接收键盘输入。当文本框获得焦点 (在包含多个可编辑的文本框的用户界面中) 时, 其中将有一个闪烁的光标。文本将被输入到该组件中。

焦点适用于任何可接受输入的组件。获得焦点的按钮有一个虚线框。

要让组件获得焦点, 可调用其 requestFocus() 方法, 且不提供任何参数, 如下例所示:

```
JButton ok = new JButton("OK");
ok.requestFocus();
```

要处理焦点事件, 类必须实现接口 FocusListener。该接口包含两个方法: focusGained(FocusEvent) 和 focusLost(FocusEvent)。它们的格式如下:

```

public void focusGained(FocusEvent event) {
    // ...
}

public void focusLost(FocusEvent event) {
    // ...
}

```

要判断哪个对象获得或失去了焦点，可调用 `FocusEvent` 对象的 `getSource()` 方法，该对象是传递给方法 `focusGained()` 和 `focusLost()` 的参数。

程序清单 12.2 中的 Java 应用程序显示两个数的和。这里使用了焦点事件来判断是否需要重新计算两个数的和。

#### 程序清单 12.2 完整的 Calculator.java 源代码

```

1: import java.awt.event.*;
2: import javax.swing.*;
3: import java.awt.*;
4:
5: public class Calculator extends JFrame implements FocusListener {
6:     JTextField value1 = new JTextField("0", 5);
7:     JLabel plus = new JLabel("+");
8:     JTextField value2 = new JTextField("0", 5);
9:     JLabel equals = new JLabel("=");
10:    JTextField sum = new JTextField("0", 5);
11:
12:    public Calculator() {
13:        super("Add Two Numbers");
14:        setSize(350, 90);
15:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16:        FlowLayout flow = new FlowLayout(FlowLayout.CENTER);
17:        setLayout(flow);
18:        // add listeners
19:        value1.addFocusListener(this);
20:        value2.addFocusListener(this);
21:        // set up sum field
22:        sum.setEditable(false);
23:        // add components
24:        add(value1);
25:        add(plus);
26:        add(value2);
27:        add(equals);
28:        add(sum);
29:        setVisible(true);
30:    }
31:
32:    public void focusGained(FocusEvent event) {
33:        try {
34:            float total = Float.parseFloat(value1.getText()) +
35:                Float.parseFloat(value2.getText());
36:            sum.setText("" + total);
37:        } catch (NumberFormatException nfe) {
38:            value1.setText("0");
39:            value2.setText("0");
40:            sum.setText("0");
41:        }
42:    }
43:
44:    public void focusLost(FocusEvent event) {
45:        focusGained(event);
46:    }
47:
48:    public static void main(String[] arguments) {
49:        Calculator frame = new Calculator();
50:    }
51: }

```

图 12.2 是这个应用程序的运行情况。

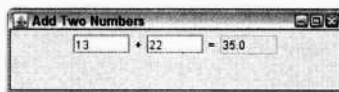


图 12.2 应用程序 Calculator

在应用程序 Calculator 中，将焦点监听器加入到了前两个文本框 value1 和 value2 中，而类实现了接口 FocusListener。

每当这两个文本框中的任何一个获得输入焦点时，都调用方法 focusGained() (第 32~42 行)。在这个方法中，将其他两个文本框中的值相加以计算它们的和。如果其中任何一个文本框中包含的值无效（如字符串），将引发 NumberFormatException 异常，并将 3 个文本框的值都重置为 0。

方法 focusLost() 调用方法 focusGained()，并将焦点事件作为参数，从而完成相同的工作。

有关这个应用程序，需要指出的一点是，为收集文本框中的数字输入，并不需要事件处理行为。所有接受文本输入的组件都将自动收集输入的值。

### 12.2.3 选项事件

选项事件在下述组件中的选项被选中或取消选中时发生：按钮、复选框和单选按钮。要处理这些事件，类必须实现接口 ItemListener。

ItemStateChanged(ItemEvent) 是接口 ItemEvent 中唯一的一个方法，其格式如下：

```
void itemStateChanged(ItemEvent event) {
    // ...
}
```

要确定事件发生在哪个选项上，可以调用 ItemEvent 对象的 getItem() 方法。

还可以使用方法 getStateChange() 来判断选项是被选中还是被取消选中。该方法返回一个整数值，它等于类变量 ItemEvent.DESELECTED 或 ItemEvent.SELECTED。

程序清单 12.3 演示了如何使用选项事件。应用程序 FormatChooser 将组合框中被选中的选项显示在一个标签中。

程序清单 12.3 完整的 FormatChooser.java 源代码

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class FormatChooser extends JFrame implements ItemListener {
6:     String[] formats = { "(choose format)", "Atom", "RSS 0.92",
7:         "RSS 1.0", "RSS 2.0" };
8:     String[] descriptions = {
9:         "Atom weblog and syndication format",
10:        "RSS syndication format 0.92 (Netscape)",
11:        "RSS/RDF syndication format 1.0 (RSS/RDF)",
12:        "RSS syndication format 2.0 (UserLand)"
13:    };
14:     JComboBox formatBox = new JComboBox();
15:     JLabel descriptionLabel = new JLabel("");
16:
17:     public FormatChooser() {
18:         super("Syndication Format");
19:         setSize(420, 150);
20:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21:         setLayout(new BorderLayout());
22:         for (int i = 0; i < formats.length; i++) {
23:             formatBox.addItem(formats[i]);
24:         }
25:         formatBox.addItemListener(this);
26:         add(BorderLayout.NORTH, formatBox);
27:         add(BorderLayout.CENTER, descriptionLabel);
28:     }
29: }
```

```

28:     setVisible(true);
29: }
30:
31: public void itemStateChanged(ItemEvent event) {
32:     int choice = formatBox.getSelectedIndex();
33:     if (choice > 0) {
34:         descriptionLabel.setText(descriptions[choice-1]);
35:     }
36: }
37:
38: public Insets getInsets() {
39:     return new Insets(50, 10, 10, 10);
40: }
41:
42: public static void main(String[] arguments) {
43:     FormatChooser fc = new FormatChooser();
44: }
45: }

```

这个应用程序扩展了第 9 章中的组合框示例。

图 12.3 显示该应用程序的运行情况。

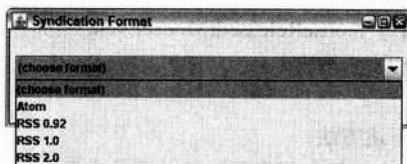


图 12.3 应用程序 FormatChooser 的输出

该应用程序使用一个字符串数组创建了一个组合框，并将一个选项监听器加入到这个组件中（第 22~25 行）。方法 `itemStateChanged(ItemEvent)`（第 31~36 行）接收选项事件，它根据被选中的选项的索引值相应地修改标签上的文本。索引值 1 对应于“Atom”，2 对应于“RSS 0.92”，3 对应于“RSS 1.0”，而 4 对应于“RSS 2.0”。

### 12.2.4 键盘事件

键盘事件在键盘上的键被按下时发生。任何组件都可以激发这些事件，要支持这些事件，类必须实现接口 `KeyListener`。

接口 `KeyListener` 中有 3 个方法：`keyPressed(KeyEvent)`、`KeyReleased(KeyEvent)` 和 `KeyTyped(KeyEvent)`。它们的格式如下：

```

public void keyPressed(KeyEvent event) {
    // ...
}

public void keyReleased(KeyEvent event) {
    // ...
}

public void keyTyped(KeyEvent event) {
    // ...
}

```

`KeyEvent` 的方法 `getKeyChar()` 返回与事件相关联的键盘字符。如果没有与按下的键对应的 Unicode 字符，`getKeyChar()` 将返回一个等于类变量 `KeyEvent.CHAR_UNDEFINED` 的字符值。

组件必须能够接受输入焦点才能引发键盘事件。文本框、文本区域以及其他能够接受键盘输入的组件自动支持这种事件。对于其他组件，如标签和面板，应使用参数 `true` 调用方法 `setFocusable(boolean)`：

```
Container pane = getContentPane();
pane.setFocusable(true);
```

### 12.2.5 鼠标事件

鼠标事件是由以下几种用户交互激发的：

- 鼠标单击；
- 鼠标进入组件区域；
- 鼠标离开组件区域。

任何组件都可以激发这些事件，类通过接口 `MouseListener` 来实现这些事件。该接口有 5 个方法：

- `mouseClicked(MouseEvent)`；
- `mouseEntered(MouseEvent)`；
- `mouseExited(MouseEvent)`；
- `mousePressed(MouseEvent)`；
- `mouseReleased(MouseEvent)`。

其中每种方法的基本格式都与 `mouseReleased(MouseEvent)` 相同：

```
public void mouseReleased(MouseEvent event) {
    // ...
}
```

可对 `MouseEvent` 对象调用下述方法。

- `getClickCount()`：返回一个整数，指出鼠标被单击的次数。
- `getPoint()`：返回一个 `Point` 对象，指出单击位置在组件中的(x, y)坐标。
- `getX()`：返回单击位置的 x 坐标。
- `getY()`：返回单击位置的 y 坐标。

### 12.2.6 鼠标移动事件

鼠标移动事件在鼠标经过组件时发生。与其他鼠标事件一样，任何组件也都可以激发鼠标移动事件。要支持这种事件，类必须实现接口 `MouseMotionListener`。

`MouseMotionListener` 接口有两个方法：`mouseDragged(MouseEvent)`和 `mouseMoved(MouseEvent)`，它们的格式如下：

```
public void mouseDragged(MouseEvent event) {
    // ...
}

public void mouseMoved(MouseEvent event) {
    // ...
}
```

与前面介绍的其他事件监听器接口不同，`MouseMotionListener` 并没有自己的事件类型，它使用的是 `MouseEvent` 对象。

因此，您可以调用与鼠标事件相同的方法：`getClick()`、`getPoint()`、`getX()`和 `getY()`。

接下来的项目演示如何检测和响应鼠标事件。程序清单 12.4 包含 `MousePrank` 和 `PrankPanel` 类，它们实现了一种流行的用户界面恶作剧：一个让用户单击不到的按钮。

#### 程序清单 12.4 完整的 `MousePrank.java` 源代码

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
```

```

4:
5: public class MousePrank extends JFrame implements ActionListener {
6:     public MousePrank() {
7:         super("Message");
8:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9:         setSize(420, 220);
10:        BorderLayout border = new BorderLayout();
11:        setLayout(border);
12:        JLabel message = new JLabel("Click OK to close this program.");
13:        add(BorderLayout.NORTH, message);
14:        PrankPanel prank = new PrankPanel();
15:        prank.ok.addActionListener(this);
16:        add(BorderLayout.CENTER, prank);
17:        setVisible(true);
18:    }
19:
20:    public void actionPerformed(ActionEvent event) {
21:        System.exit(0);
22:    }
23:
24:    public Insets getInsets() {
25:        return new Insets(40, 10, 10, 10);
26:    }
27:
28:    public static void main(String[] arguments) {
29:        new MousePrank();
30:    }
31: }
32:
33: class PrankPanel extends JPanel implements MouseMotionListener {
34:     JButton ok = new JButton("OK");
35:     int buttonX, buttonY, mouseX, mouseY;
36:     int width, height;
37:
38:     PrankPanel() {
39:         super();
40:         setLayout(null);
41:         addMouseMotionListener(this);
42:         buttonX = 110;
43:         buttonY = 110;
44:         ok.setBounds(new Rectangle(buttonX, buttonY,
45:             70, 20));
46:         add(ok);
47:     }
48:
49:     public void mouseMoved(MouseEvent event) {
50:         mouseX = event.getX();
51:         mouseY = event.getY();
52:         width = (int)getSize().getWidth();
53:         height = (int)getSize().getHeight();
54:         if (Math.abs((mouseX + 35) - buttonX) < 50) {
55:             buttonX = moveButton(mouseX, buttonX, width);
56:             repaint();
57:         }
58:         if (Math.abs((mouseY + 10) - buttonY) < 50) {
59:             buttonY = moveButton(mouseY, buttonY, height);
60:             repaint();
61:         }
62:     }
63:
64:     public void mouseDragged(MouseEvent event) {
65:         // ignore this event
66:     }
67:
68:     private int moveButton(int mouseAt, int buttonAt, int border) {
69:         if (buttonAt < mouseAt) {
70:             buttonAt--;
71:         } else {
72:             buttonAt++;
73:         }
74:         if (buttonAt > (border - 20)) {

```



```

75:         buttonAt = 10;
76:     }
77:     if (buttonAt < 0) {
78:         buttonAt = border - 80;
79:     }
80:     return buttonAt;
81: }
82:
83: public void paintComponent(Graphics comp) {
84:     super.paintComponent(comp);
85:     ok.setBounds(buttonX, buttonY, 70, 20);
86: }
87: }

```

MousePrank 类是一个框架，它容纳两个以边框布局方式排列的组件：标签“Click OK to close this program”和一个上面有一个 OK 按钮的面板。图 12.4 显示了这个应用程序的用户界面。

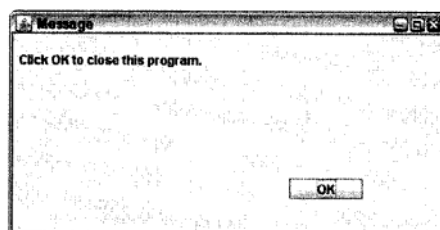


图 12.4 应用程序 MousePrank

由于这个按钮不以正常方式运行，因此是使用 JPanel 类的子类 PrankPanel 实现的。这个面板中包含一个按钮，按钮被绘制在面板中的特定位置，而不是使用布局管理器放置的。这种技术在第 11 章的末尾介绍过。

首先将面板的布局管理器设置为 null，这样它将不会使用默认的顺序布局：

```
setLayout(null);
```

接下来，使用 setBounds(Rectangle) 将按钮放置在面板上，这个方法也用于指定框架或窗口将出现在桌面的什么位置。

Rectangle 对象是使用 4 个参数创建的：x 位置、y 位置、宽度和高度。下面的代码演示了 PrankPanel 是如何绘制按钮的：

```

JButton ok = new JButton("OK");
int buttonX = 110;
int buttonY = 110;
ok.setBounds(new Rectangle(buttonX, buttonY, 70, 20));

```

在方法调用中创建 Rectangle 对象的效率将更高，因为不需要在这个类的其他地方使用它。下面的代码分两步完成了同样的任务：

```

Rectangle box = new Rectangle(buttonX, buttonY, 70, 20);
ok.setBounds(box);

```

这个类有两个用于存储按钮的 x 位置和 y 位置的实例变量：buttonx 和 buttony。它们的初始值都是 110，但当鼠标离按钮中心的距离不超过 50 像素时，它们的值将被修改。

鼠标移动是通过实现接口 MouseListener 及其两个方法——mouseMoved(MouseEvent) 和 mouseDragged(MouseEvent)——来跟踪的。

面板使用了方法 mouseMoved()，而忽略了 mouseDragged()。

鼠标移动时，鼠标事件对象的 getX() 和 getY() 方法将返回它的 x 位置和 y 位置，它们存储在实例变量 mouseX 和 mouseY 中。

方法 `moveButton(int, int, int)` 接受 3 个参数：

- 按钮的 x 位置或 y 位置；
- 鼠标的 x 位置或 y 位置；
- 面板的宽度或高度。

这个方法沿水平或垂直方向移动按钮，使之远离鼠标；具体沿哪个方向移动取决于调用它时指定的参数是 x 坐标和面板高度还是 y 坐标和面板宽度。

移动按钮的位置后，调用方法 `repaint()`，这将导致面板的方法 `paintComponent(Graphics)`（第 83~86 行）被调用。

每个组件都有方法 `paintComponent()`，可以对其进行覆盖以绘制组件。按钮的 `setBounds()` 方法在当前的(x, y)处显示按钮（第 85 行）。

### 12.2.7 窗口事件

窗口事件在用户打开或关闭诸如 `JFrame` 或 `JWindow` 等窗口对象时发生。任何组件都可以激发这种事件，要支持它们，类必须实现接口 `WindowListener`。

接口 `WindowListener` 有 7 个方法：

- `windowActivated(WindowEvent)`;
- `windowClosed(WindowEvent)`;
- `windowClosing(WindowEvent)`;
- `windowDeactivated(WindowEvent)`;
- `windowDeiconified(WindowEvent)`;
- `windowIconified(WindowEvent)`;
- `windowOpened(WindowEvent)`。

它们的格式相同，下面是 `WindowOpened()` 的格式：

```
public void windowOpened(WindowEvent evt) {
    // ...
}
```

方法 `windowClosing()` 和 `windowClosed()` 极其相似，但前者在窗口正关闭时调用，后者在关闭后调用。实际上，可以在方法 `windowClosing()` 中采取某种措施来阻止窗口被关闭。

还有一个名为 `WindowAdapter` 的适配器（adapter）类，它实现了 `WindowListener` 接口。第 9 章中的 `ExitWindow` 应用程序使用了这个类。

### 12.2.8 使用适配器类

Java 类实现接口时，必须包括该接口的所有方法，即使不打算使用这些方法。

这种要求使得在实现事件处理接口（如 `WindowListener`，它有 7 个方法）时，必须添加大量的空方法。

出于方便的考虑，Java 提供了适配器（adapter）——包含特定接口的空实现的 Java 类。通过继承适配器类，可以只实现需要的事件处理方法——覆盖这些方法，其他的方法将继承超类中的空方法。

`java.awt.event` 包提供了 `FocusAdapter`、`KeyAdapter`、`MouseAdapter`、`MouseMotionAdapter` 和 `WindowAdapter` 等适配器类，它们分别对应于用于焦点、键盘、鼠标移动和窗口等事件的监听器。

程序清单 12.5 中的应用程序通过一个 `KeyAdapter` 的子类来监控键盘事件，进而显示用户最后一次按下的键。

程序清单 12.5 完整的 KeyChecker.java 源代码

```

1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class KeyChecker extends JFrame {
6:     JLabel keyLabel = new JLabel("Hit any key");
7:
8:     public KeyChecker() {
9:         super("Hit a Key");
10:        setSize(300, 200);
11:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:        setLayout(new FlowLayout(FlowLayout.CENTER));
13:        KeyMonitor monitor = new KeyMonitor(this);
14:        setFocusable(true);
15:        addKeyListener(monitor);
16:        add(keyLabel);
17:        setVisible(true);
18:    }
19:
20:    public static void main(String[] arguments) {
21:        new KeyChecker();
22:    }
23: }
24:
25: class KeyMonitor extends KeyAdapter {
26:     KeyChecker display;
27:
28:     KeyMonitor(KeyChecker display) {
29:         this.display = display;
30:     }
31:
32:     public void keyTyped(KeyEvent event) {
33:         display.keyLabel.setText("" + event.getKeyChar());
34:         display.repaint();
35:     }
36: }

```

## 12.3 总结

将 Swing 的事件处理系统加入到程序中的步骤都相同：

- 在将包含事件处理方法的类中实现监听器接口；
- 将监听器加入到每个将激发要处理事件的组件中；
- 加入方法，其中每个方法以 EventObject 作为其唯一的参数；
- 使用 EventObject 类的方法（如 getSource）来获悉事件的类型以及激发该事件的组件。

知道这些步骤后，就可以使用任何监听器接口和事件类。

## 12.4 问与答

问：可将程序的事件处理行为放在独立的类中，而不将它放在创建接口的代码中吗？

答：可以，很多程序员都认为这是一种设计程序的好办法。将界面设计与事件处理代码分开使得可以分别开发它们。本章的应用程序 SwingColorTest 采用的是另一种方式，这使得工程更容易维护，相关的行为被组织在一起，并与无关的行为分开。

问：有办法判断出单击的是哪个鼠标按钮吗？

答：可以，用一种鼠标事件特性。本章中并没有介绍这种特性，这是因为右按钮和中按钮特性是平台特定的，并非运行 Java 程序的所有系统上都有。

所有的鼠标事件都将一个 `MouseEvent` 对象传递给它的事件处理方法。调用该对象的 `getModifiers()` 方法可获得一个整数值，它指出了事件是由哪个鼠标按钮激发的。

将这个值同 3 个类变量进行比较。如果等于 `MouseEvent.BUTTON1_MASK`，则说明单击的是左按钮；如果是 `MouseEvent.BUTTON2_MASK`，则单击的是中按钮；如果是 `MouseEvent.BUTTON3_MASK`，则单击的是右按钮。

## 12.5 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 12.5.1 问题

- 如果在方法调用中使用 `this`（如 `addActionListener(this)`），哪个对象将被注册为监听器？
  - 适配器（adapter）类；
  - 当前类；
  - 没有。
- 继承诸如 `WindowAdapter`（它实现了 `WindowListener` 接口）等适配器类有何好处？
  - 将继承这个类的所有行为；
  - 子类将自动成为监听器；
  - 不必实现任何不需要的 `WindowListener` 方法。
- 当您按下 `Tab` 键以离开文本框时，将激发什么事件？
  - `FocusEvent`；
  - `WindowEvent`；
  - `ActionEvent`。

#### 答案

- b，当前类必须实现正确的监听器接口和所需的方法。
- c，由于大多数监听器接口都包含了您可能不必要的方法，使用适配器类作为超类，可避免为实现接口而必须实现一些空方法的麻烦。
- a，当用户停止编辑组件，并移到界面的其他组件中时，该组件将失去焦点。

### 12.5.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

对于下述代码：

```
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;

public class Interface extends JFrame implements ActionListener {
    public boolean deleteFile;

    public Interface() {
        super("Interface");
        JLabel commandLabel = new JLabel("Do you want to delete the file?");
        JButton yes = new JButton("Yes");
        JButton no = new JButton("No");
        yes.addActionListener(this);
    }
}
```

```

        no.addActionListener(this);
setLayout( new BorderLayout() );
JPanel bottom = new JPanel();
bottom.add(yes);
bottom.add(no);
add("North", commandLabel);
add("South", bottom);
pack();
setVisible(true);
}

public void actionPerformed(ActionEvent evt) {
    JButton source = (JButton) evt.getSource();
    // answer goes here
    deleteFile = true;
    else
        deleteFile = false;
}

public static void main(String[] arguments) {
    new Interface();
}
}

```

要使其正确运行，应将// answer goes here 替换为下述哪条语句？

- a. if (source instanceof JButton)
- b. if (source.getActionCommand().equals("yes"))
- c. if (source.getActionCommand().equals("Yes"))
- d. if source.getActionCommand() == "Yes"

答案 c

## 12.6 练习

为巩固本章介绍的知识，请尝试完成下面的练习：

1. 创建一个应用程序，使用 `FocusListener` 来确保当用户将文本框的值修改为负数时，将其乘以-1，并将结果重新显示到文本框中。
2. 创建简单的计算器，当用户单击其中的按钮时，将两个文本框的内容相加或相减，并将结果显示在标签中。



## 第 13 章

# 使用颜色、字体和图形

本章介绍用于将图形加入到图形用户界面中的 Java 类，为此需要使用 Java2D——一组支持高质量二维图形、图像、颜色和文本的类。

Java2D 包含 Java 类库中 `java.awt` 和 `javax.swing` 包中的类，可用于绘制诸如圆、多边形等几何形状以及文本，使用各种字体、颜色和线宽，处理颜色和图案。

### 13.1 Graphics2D 类

Java2D 的一切都基于 `java.awt` 包中的 `Graphics2D` 类，后者表示一种可在其中绘制图形的环境。`Graphics2D` 对象可能表示图形用户界面、打印机或其他显示设备上的一个组件。

`Graphics2D` 是 `Graphics` 的子类，并包含 Java2D 所需的扩展特性。

#### 注意

在以前的 Java 版本中，`Graphics` 类包含基本的图形支持功能，但在 Java2D 中，这些方法已被更复杂、更高效的方法所代替。

有多种用户界面组件可用作图形操作的画布：面板、窗口。

有了可用作画布的界面组件后，便可以在该对象上绘制文本、直线、椭圆、圆、圆弧以及矩形和其他多边形。

适合这种用途的组件之一是 `javax.swing` 包中的 `JPanel`。这个类表示图形用户界面中的面板，它可以为空，也可以包含其他组件。

下面的例子创建了一个框架和一个面板，并将面板加入到框架中：

```
JFrame main = new JFrame("Main Menu");
JPanel pane = new JPanel();
Container content = main.getContentPane();
content.add(pane);
```

框架的 `getContentPane()` 方法返回一个 `Container` 对象，该对象表示框架中可放置其他组件的部分。为将面板加入到框架中，调用了该容器对象的 `add()` 方法。

像 Java 中很多其他的用户界面组件一样，`JPanel` 对象也有一个 `paintComponent(Graphics)` 方法，每当组件需要重新显示时，该方法都将被自动调用。

导致 `paintComponent()` 被调用的事件很多，其中包括：

- 包含组件的图形用户界面首次被显示；
- 位于组件上面的窗口被关闭；
- 包含组件的图形用户界面的大小被调整。

通过创建 `JPanel` 的子类，可以覆盖面板的 `paintComponent()` 方法，并将所有的图形操作代码放在这个方法中。

您可能注意到了，组件的 `paintComponent()` 接受一个 `Graphics` 对象（而不是 `Graphics2D` 对象）作为参数。要创建表示组件的绘制表面的 `Graphics2D` 对象，必须将其进行强制类型转换，如下所示：

```
public void paintComponent(Graphics comp) {
    Graphics2D comp2D = (Graphics2D)comp;
    // ...
}
```

这里的 `comp2D` 对象是通过强制类型转换得到的。

### 13.1.1 图形坐标系

`Java2D` 类使用的  $x, y$  坐标系与您设置框架和其他组件的大小时使用的相同。

Java 的坐标系以像素为度量单位。坐标原点  $(0, 0)$  位于组件的左上角。

从原点沿水平方向向右移动时， $x$  坐标将增大；沿垂直方向向下移动时， $y$  坐标将增大。

当您调用方法 `setSize(int, int)` 设置框架的大小时，框架的左上角坐标为  $(0, 0)$ ，右下角的坐标为传递给 `setSize()` 的两个参数值。

例如，下面的语句创建了一个宽和高分别为 425 和 130 个像素的框架，其右下角的坐标为  $(425, 130)$ 。  
`setSize(425, 130);`

#### 警告

这不同于其他绘图系统，在这些系统中，原点位于左下角， $y$  坐标沿垂直方向向上增加。

所有的像素值都是整数，不能使用小数来指定显示位置。

图 13.1 说明了 Java 的图形坐标系，其中显示了原点  $(0,0)$  和矩形的两个顶点（坐标分别为  $(20, 20)$  和  $(60, 60)$ ）。

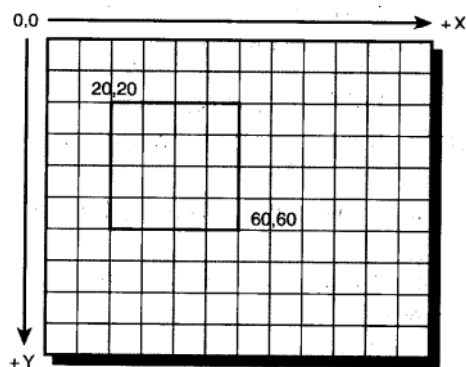


图 13.1 Java 的图形坐标系

### 13.1.2 绘制文本

在界面组件上，最容易绘制的是文本。

要绘制文本，可调用 `Graphics2D` 对象的 `drawString()` 方法，并提供下述 3 个参数：

- 要显示的 `String`；
- 显示位置的  $x$  坐标；
- 显示位置的  $y$  坐标。

`drawString()` 方法中使用的  $x$  坐标和  $y$  坐标指的是字符串左下角的像素。

下面的 `paintComponent()` 方法在坐标  $(22, 100)$  处绘制字符串 “Free the bound periodicals.”：

```
public void paintComponent(Graphics comp) {
    Graphics2D comp2D = (Graphics2D)comp;
    comp2D.drawString("Free the bound periodicals", 22, 100);
}
```

上述范例使用的是默认字体。要使用其他字体，必须创建一个 Font 对象。Font 类位于 java.awt 包中。

Font 对象表示字体的名称、字形和字号。

创建 Font 对象时，需要将 3 个参数传递给它的构造函数：

- 字体名称；
- 字形；
- 字号。

字体名可以是字体的逻辑名称，如 Arial、Courier New、Garamond 或 Kaiser。如果运行 Java 程序的系统有指定的字体，将使用这种字体；如果没有，则使用默认字体。

字体名也可以是 5 种通用字体之一：Dialog、DialogInput、Monospaced、SanSerif 或 Serif。这些字体可用来指定要使用的字体类型，而不要求使用特定的字体。这通常是一种更好的选择，因为有些字体并非在所有的 Java 实现中都有。

可选择的字形有 3 种，它们分别是静态类常量 Font.PLAIN、Font.BOLD 和 Font.ITALIC。这些常量都是整数，您可以将它们相加来实现多种字形。

下面的语句创建一个 24 磅的 Dialog 字体，其字形为粗体和斜体：

```
Font f = new Font("Dialog", Font.BOLD + Font.ITALIC, 24);
```

创建字体后，便可以以该字体为参数调用 Graphics2D 的 setFont(Font)方法来使用它。

setFont()方法设置后面调用 drawString()方法时将使用的字体。以后绘制其他文本时，可以再次调用 setFont()方法来修改字体。

下面的 paintComponent()方法创建了一个新的 Font 对象，并使用这种字体在坐标 (10,100) 处绘制字符串 "I'm very font of you."。

```
public void paintComponent(Graphics comp) {
    Graphics2D comp2D = (Graphics2D)comp;
    Font f = new Font("Arial Narrow", Font.PLAIN, 72);
    comp2D.setFont(f);
    comp2D.drawString("I'm very font of you", 10, 100);
}
```

可在 Java 程序中包含并加载文件中的字体来确保它可用。为此，需要使用 Font 类的 Create(int, InputStream)方法，它返回一个表示字体的 Font 对象。

输入流将在第 15 章介绍，它们是能够从诸如磁盘文件或 Web 地址等信息源加载数据的对象。下面的语句加载文件 Verdana.ttf 中的一种字体，该文件与这些语句所属的类文件位于同一个文件夹中：

```
try {
    File ttf = new File("Verdana.ttf");
    FileInputStream fis = new FileInputStream(ttf);
    Font font = Font.createFont(Font.TRUETYPE_FONT, fis);
} catch (IOException ioe) {
    System.out.println("Error: " + ioe.getMessage());
    ioe.printStackTrace();
}
```

try-catch 块处理输入/输出错误，从文件中加载数据时必须这样做。类 File、FileInputStream 和 IOException 位于 java.io 包中，将在第 15 章介绍。

使用 createFont()加载字体时，字体的大小为 1 磅，字形为“常规”。要修改大小和字形，可调用字体对象的 deriveFont(int, int)方法，并使用所需的字形和大小作为参数。



### 13.1.3 通过反走样改善字体和图形的质量

使用前面介绍的技巧来显示文本时，字体的外观与其他软件显示的字体相比，显得极其粗糙。显示的字符边缘呈锯齿状，在曲线和斜线中，这尤其明显。

通过使用反走样功能，Java2D 能够绘制出质量更高的字体和图形。反走样是一种这样的技术：通过修改像素周围的颜色，使粗造的边缘更为平滑。

默认情况下，这种功能被关闭。要启用它，可使用下述两个参数调用 Graphics2D 对象的 `setRenderingHint()` 方法：

- 指定要设置的渲染参数 (rendering hint) 的 `RenderingHint.Key` 对象；
- 设置该参数值的 `RenderingHint.Key` 对象。

下面的代码在一个名为 `comp2D` 的 `Graphic2D` 对象中启用反走样功能：

```
comp2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
```

通过在组件的 `paintComponent()` 方法中调用上述方法，可使接下来的所有绘图操作都使用反走样功能。

### 13.1.4 获取字体的信息

为使图形用户界面中的文本美观大方，常常需要确定文本在界面组件上占用多少空间。

`java.awt` 包中的 `FontMetrics` 类提供了这样的方法，即可以确定以指定的字体显示时字符有多大。这些方法可用于实现格式化或文本居中等。

`FontMetrics` 类可用于获取有关当前字体的详细信息，如字符的宽度和高度。

要使用这个类的方法，必须调用 `getFontMetrics()` 方法来创建一个 `FontMetrics` 对象。该方法接受一个参数：`Font` 对象。

表 13.1 列出了使用 `FontMetrics` 可获取的一些信息。所有这些方法都必须对 `FontMetrics` 对象进行调用。

表 13.1 FontMetrics 的方法

方法名	操作
<code>stringWidth(String)</code>	对于给定的字符串，返回其宽度（单位为像素）
<code>charWidth(char)</code>	对于给定的字符，返回其宽度
<code>getHeight()</code>	返回字体的高度

程序清单 13.1 演示了如何使用 `Font` 和 `FontMetrics` 类。应用程序 `TextFrame` 在框架的中央显示一个字符串，并使用 `FontMetrics` 来确定使用当前字体时该字符串的宽度。

程序清单 13.1 完整的 `TextFrame.java` 源代码

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class TextFrame extends JFrame {
6:     public TextFrame(String text, String fontName) {
7:         super("Show Font");
8:         setSize(425, 150);
9:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10:        TextFramePanel sf = new TextFramePanel(text, fontName);
11:        add(sf);
12:        setVisible(true);
13:    }
14: }
```

```

13:     }
14:
15:     public static void main(String[] arguments) {
16:         if (arguments.length < 1) {
17:             System.out.println("Usage: java TextFrame message font");
18:             System.exit(-1);
19:         }
20:         TextFrame frame = new TextFrame(arguments[0], arguments[1]);
21:     }
22: }
23:
24:
25: class TextFramePanel extends JPanel {
26:     String text;
27:     String fontName;
28:
29:     public TextFramePanel(String text, String fontName) {
30:         super();
31:         this.text = text;
32:         this.fontName = fontName;
33:     }
34:
35:     public void paintComponent(Graphics comp) {
36:         super.paintComponent(comp);
37:         Graphics2D comp2D = (Graphics2D)comp;
38:         comp2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
39:             RenderingHints.VALUE_ANTIALIAS_ON);
40:         Font font = new Font(fontName, Font.BOLD, 18);
41:         FontMetrics metrics = getFontMetrics(font);
42:         comp2D.setFont(font);
43:         int x = (getSize().width - metrics.stringWidth(text)) / 2;
44:         int y = getSize().height / 2;
45:         comp2D.drawString(text, x, y);
46:     }
47: }

```

应用程序 TextFrame 接受两个命令行参数：要显示的文本以及要使用的字体，如下例所示：

```
java TextFrame "Able was I ere I saw Elba" "Times New Roman"
```

图 13.2 显示了该应用程序在安装了 Times New Roman 字体的系统上的运行情况。运行该应用程序时，请调整界面的大小，看看文本如何移动，以停留在中央。

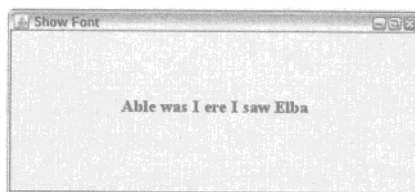


图 13.2 将文本显示在图形用户界面的中央

应用程序 TextFrame 由两个类组成：一个框架和一个名为 TextFramePanel 的面板子类。通过覆盖 paintComponent(Graphics) 方法，并在其中调用 Graphics2D 类的绘图方法，在面板上绘制文本。

第 43 行和第 44 行的 getSize() 方法根据面板的宽度和高度来确定文本的显示位置。当界面的大小被调整时，面板的大小也将被调整，导致 paintComponent() 方法被自动调用。

## 13.2 颜色

java.awt 包的 Color 和 ColorSpace 类可用于使图形用户界面更生动。有了这些类，可以设置画图操作中使用的颜色以及界面组件和其他窗口的背景色，还可以将颜色从一种颜色描述系统转换为另一种系统。

默认情况下, Java 根据颜色描述系统 sRGB 来使用颜色。在这种系统中, 颜色是用其包含的红色、绿色和蓝色成分来描述的——这正是 R、G 和 B 的意思。这 3 种组件都用 0~255 的整数表示。黑色为 (0, 0, 0), 即没有红色、绿色和蓝色; 白色为 (255, 255, 255), 即 3 种组分的值都最大。也可以用 3 个 0~1.0 的浮点数来表示 sRGB 值。使用 sRGB, Java 可以表示数百万种颜色。

颜色描述系统也叫颜色空间 (color space), sRGB 只是其中的一种颜色空间。还有一种名为 CMYK 的系统, 这是由打印机使用的, 它通过青色、品红、黄色和黑色的量来描述颜色。Java 2 支持使用任何颜色空间, 只要使用一个定义该颜色描述系统的 ColorSpace 对象即可。您还可以在任何颜色空间和 sRGB 之间进行转换。

在 Java 内部, 使用 sRGB 来表示颜色, 这只是程序使用的颜色空间之一。诸如显示器或打印机等输出设备有自己的颜色空间。

当您显示或打印特定颜色的内容时, 输出设备可能不支持这种颜色。在这种情况下, 将使用其他颜色来代替, 或者使用抖动 (dithering) 图案来近似表示不可用的颜色。在万维网上, 这种现象经常发生: 当某种颜色没有时, 使用与之相近的两种或多种颜色抖动图案来代替。

颜色管理的现实意义在于, 您指定的 sRGB 颜色并不是在所有的输出设备上都可用。如果需要更精确地控制颜色, 可以使用 java.awt.color 包中的 ColorSpace 类或其他类。

对大多数程序而言, 使用内置的 sRGB 来定义颜色足够了。

### 13.2.1 使用 Color 对象

颜色是用 Color 对象表示的。要创建这种对象, 可使用构造函数, 也可以使用 Color 类中的标准颜色。

为创建颜色, 可通过两种不同的方式来调用 Color 构造函数:

- 使用 3 个表示所需颜色的 sRGB 值的整数;
- 使用 3 个表示所需颜色的 sRGB 值的浮点数。

可以使用 int 或 float 值来指定颜色的 sRGB 值。下面的语句演示了这两种情况:

```
Color c1 = new Color(0.807F, 1F, 0F);
```

```
Color c2 = new Color(255, 204, 102);
```

对象 c1 是霓虹绿, 而 c2 是奶油色。

#### 注意

诸如 0F 和 1F 等浮点字面量很容易与十六进制数混淆, 这些内容在第 2 章中讨论过了。颜色通常表示为十六进制数, 例如, 使用 HTML BODY 标签来设置网页的背景色时。您使用的 Java 类和方法不接受十六进制的参数, 因此当您看到诸如 1F 或 0F 等字面量时, 处理的是浮点数。

### 13.2.2 检测和设置当前颜色

当前的绘图颜色可使用类 Graphics 的 setColor() 方法来指定。必须对表示绘制区域的 Graphics 或 Graphics2D 对象调用该方法。

类 Color 中包含一些表示最常用颜色的类变量。

表示颜色的 Color 类变量如下 (括号中是颜色的 sRGB 值):

black (0, 0, 0)	magenta (255, 0, 255)
blue (0, 0, 255)	orange (255, 200, 0)
cyan (0, 255, 255)	pink (255, 175, 175)

```

darkGray (64, 64, 64)          red (255, 0, 0)
gray (128, 128, 128)          white (255, 255, 255)
green (0, 255, 0)              yellow (255, 255, 0)
lightGray (192, 192, 192)

```

下面的语句使用一个标准类变量来设置一个名为 comp2D 的 Graphic2D 对象的颜色：

```
comp2D.setColor(Color.pink);
```

如果创建了 Color 对象，可以使用它以同样的方式来设置颜色：

```

Color brush = new Color(255, 204, 102);
comp2D.setColor(brush);

```

设置当前颜色后，接下来的所有画图操作都将使用这种颜色。

您可以设置诸如面板和框架等组件的背景色，方法是调用该组件的 setBackground() 和 setForeground() 方法。

方法 setBackground() 设置组件的背景色，如下例所示：

```
setBackground(Color.white);
```

还有一个 setForeground() 方法，可以对用户界面组件调用该方法，它修改诸如按钮和窗口等界面组件的颜色。

可以在方法 init() 中调用 setForeground() 来设置画图操作的颜色。这种颜色将一直被使用，直到调用 setForeground() 或 setColor() 来选择了另一种颜色。

要知道当前的颜色，对于 Graphics2D 对象，可调用 getColor() 方法；对于组件，可调用 getForeground() 或 getBackground() 方法。

下面的语句将名为 comp2D 的 Graphics2D 对象的当前颜色设置为一个组件的背景色：

```
comp2D.setColor(getBackground());
```

## 13.3 绘制直线和多边形

本章介绍的所有基本绘图命令都是 Graphics2D 方法，这些方法将在组件的 paintComponent() 方法中调用。

所有的绘图操作都适合在这个方法中完成，因为每当组件需要重新显示时，paintComponent() 方法都将自动被调用。

如果组件被另一个程序的窗口覆盖，从而需要重新绘制，则将绘制操作放在 paintComponent() 方法中可确保不会遗漏任何一部分。

Java2D 特性包括：

- 绘制空的多边形或用纯色填充的多边形；
- 使用特殊的填充图案，如渐变和图案；
- 定义了画笔的宽度和样式；
- 通过反走样来使对象的边缘平滑化。

### 13.3.1 用户和设备坐标空间

Java2D 引入的概念之一是，输出设备坐标空间与您在绘制对象时参考的坐标空间不同。

坐标空间 (Coordinate space) 是可以用 (x, y) 坐标来描述的 2D 区域。

在 Java 2 之前，所有的绘图操作使用的都是设备坐标空间。您可以指定输出表面（如面板）的 (x, y) 坐标，该坐标将用于绘制文本和其他元素。

创建对象并实际绘制它时，Java2D 要求参考另一种坐标空间。这被称为用户坐标空间。

在程序中进行 2D 绘制操作之前，设备空间和用户空间的原点是重叠的，都位于画图区域的左上角。

执行 2D 绘图操作后，用户空间的原点可能移动。甚至  $x$  轴和  $y$  轴也可能因为 2D 旋转而移动。使用 Java2D 时，您将更详细地了解这两种坐标系。

### 13.3.2 指定渲染属性

在 2D 绘图的下一步是指定绘制的对象将如何被渲染。绘制非 2D 对象时，只能选择一个属性：颜色。Java2D 提供了大量的属性，用于指定颜色、线宽、填充图案、透明度和其他特征。

#### 1. 填充图案

填充图案控制对象将如何被填充。在 Java2D 中，您可以使用纯色、渐变填充、纹理或您自己设计的图案。

使用 Graphics2D 的 `setPaint()` 方法可以定义填充图案，该方法接受一个 Paint 对象作为其唯一的参数。可用作填充图案的类（包括 `GradientPaint`、`TexturePaint` 和 `Color`）都可以实现接口 `Paint`。第三个类可能让您感到惊讶，但将 `Color` 对象用作 `setPaint()` 的参数与将纯色作为填充图案等效。

渐变填充（gradient fill）指的是从一个坐标点上的某种颜色逐渐变化到另一个坐标点上的另一种颜色。这种变化可在两个点间发生一次或多次，前者称为非周期性渐变（acyclic gradient），后者称为周期性渐变（cyclic gradient）。

图 13.3 显示了黑白之间的非周期渐变和周期渐变。箭头指明了发生色移（color shift）的点。

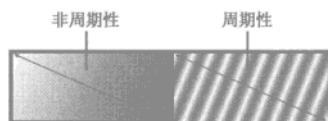


图 13.3 非周期渐变和周期渐变

在渐变中的坐标点指向的并非在其上绘制的 Graphics2D 对象上的点，它们参照的是用户空间，因此可以位于用渐变填充的对象之外。

图 13.4 说明了这一点。其中的两个矩形都用相同的 GradientPaint 对象进行填充。可以把渐变图案看作是一块在平坦表面上展开的布。用渐变填充的图形是从布上剪下的图案，且可从同一块布上剪切下多个图案。

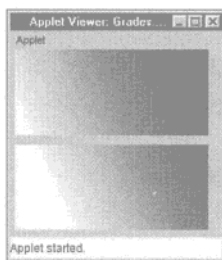


图 13.4 两个使用相同 GradientPaint 的矩形

调用 `GradientPaint` 构造方法的格式如下：

```
GradientPaint gp = new GradientPaint(
    x1, y1, color1, x2, y2, color2);
```

点  $(x1, y1)$  是渐变的起点，其颜色为 `color1`；点  $(x2, y2)$  是渐变的端点，其颜色为 `color2`。

要使用周期性渐变，可以加上另一个参数：

```
GradientPaint gp = new GradientPaint(
    x1, y1, color1, x2, y2, color2, true);
```

最后的参数是一个布尔值，为 `true` 时，表示周期性渐变；为 `false` 时，表示非周期性渐变。省略该参数时，表示非周期性渐变——这是默认行为。

创建 `GradientPaint` 对象后，可以使用方法 `setPaint()` 来将其设置为当前的绘图属性。下面的语句创建并选择了一种渐变：

```
GradientPaint pat = new GradientPaint(0f,0f,Color.white,
    100f,45f,Color.blue);
comp2D.setPaint(pat);
```

接下来在 `comp2D` 对象上执行的所有绘图操作都将使用这种填充图案，直到选择另一种填充图案为止。

## 2. 设置画笔

在 Java2D 中，可以通过调用 `setStroke()` 方法并将 `BasicStroke` 作为参数传递给它，来改变绘制的直线宽度。

简单的 `BasicStroke` 构造函数接受 3 个参数。

- 一个 `float` 值：表示线宽，通常为 1.0。
- 一个 `int` 值：决定线段两端的修饰样式。
- 一个 `int` 值：决定线段间的连接样式。

端点样式 (`endcap-style`) 和连接样式 (`juncture-style`) 参数用 `BasicStroke` 类变量指定。端点样式用于不与其他线段相连的线段的两端；连接样式用于与其他线段相连的线段的两端。

端点样式有 `CAP_BUTT` (没有端点)、`CAP_ROUND` (圆形端点) 和 `CAP_SQUARE` (方形端点)。图 13.5 显示了每种端点样式。正如您看到的，`CAP_BUTT` 和 `CAP_SQUARE` 之间的唯一差别是，`CAP_SQUARE` 更长一些，因为它加入了方形端点。



图 13.5 端点样式

连接样式有 `JOIN_MITER` (通过扩展两条线段的外边界来将它们连接起来)、`JOIN_ROUND` (将两条线段圆滑地连接起来) 和 `JOIN_BEVEL` (用一条直线将两条线段连接起来)。图 13.6 显示了每种连接样式。



图 13.6 端点连接样式

下面的语句创建一个 `BasicStroke` 对象，并将其用作当前画笔。

```
BasicStroke pen = BasicStroke(2.0f,
    BasicStroke.CAP_BUTT,
    BasicStroke.JOIN_ROUND);
comp2D.setStroke(pen);
```

该画笔的宽度为 2 像素，没有端点样式，连接样式为圆角。

### 13.3.3 创建要绘制的对象

创建 `Graphics2D` 并指定其渲染属性后，最后两步是创建要绘制的对象并绘制它。

在 Java2D 中，要绘制的对象是通过使用 `java.awt.geom` 包中的类，将其定义为几何图形来创建的。

您可以绘制线段、矩形、椭圆、圆弧和多边形。

绘制各种图形时，不是调用 Graphics2D 类的不同方法，而是定义要绘制的图形，并将其用作方法 draw() 或 fill() 的参数。

### 1. 线段

线段是使用类 Line2D.Float 来创建的。这个类接受 4 个参数：两个端点的  $x$  坐标和  $y$  坐标。下面是一个例子：

```
Line2D.Float ln = new Line2D.Float(60F, 5F, 13F, 28F);
```

该语句创建一条端点分别为(60, 5)和(13, 28)的线段。注意，这里使用 F 将字面量参数指定为 float 类型，否则，Java 编译器将认为它们是整数。

### 2. 矩形

矩形是使用类 Rectangle2D.Float 或 Rectangle2D.Double 来创建的。这两个类之间的区别在于：一个接受 float 参数，另一个接受 double 参数。

Rectangle2D.Float 接受 4 个参数： $x$  坐标、 $y$  坐标、宽度和高度。下面是一个例子：

```
Rectangle2D.Float rc = new Rectangle2D.Float(10F, 13F, 40F, 20F);
```

上述语句创建一个矩形，其左上角的坐标为(10, 13)，宽度和高度分别为 40 和 20 个像素。

### 3. 椭圆

椭圆是使用类 Ellipse2D.Float 来创建的。它接受 4 个参数： $x$  坐标、 $y$  坐标、宽度和高度。

下面的语句创建了一个椭圆，其外切矩形的左上角坐标为(113, 25)，宽度和高度分别为 22 和 40 个像素：

```
Ellipse2D.Float ee = new Ellipse2D.Float(113, 25, 22, 40);
```

### 4. 弧

在所有在 Java2D 中可以绘制的几何形状中，弧是最复杂的。

圆弧是使用 Arc2D.Float 类来创建的，它接受 7 个参数：

- 圆弧所属椭圆的外切矩形的左上角的  $x$  坐标和  $y$  坐标；
- 椭圆的宽度和高度；
- 弧的起始角度；
- 弧环绕的角度；
- 指定如何闭合的整数。

弧逆时针环绕时，用负数指定其环绕角度。

图 13.7 说明了起始角度的值是如何表示的。起始角度的取值范围为  $0 \sim 359$ 。 $0$  表示 3 点钟的位置， $90^\circ$  表示 12 点钟的位置， $180^\circ$  表示 9 点钟的位置，而  $270^\circ$  是 6 点钟的位置。

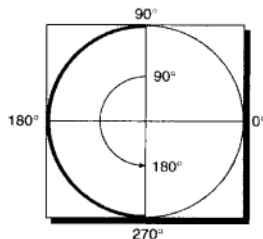


图 13.7 起始角度

最后的参数使用下列 3 个类变量之一：Arc2D.OPEN（不闭合）、Arc2D.CHORD（使用线段来连接弧的两个端点）和 Arc2D.PIE（将弧与椭圆的中心连接起来，就像扇形）。图 13.8 显示了这些闭合样式。



图 13.8 弧的闭合样式

#### 注意

闭合样式 Arc2D.OPEN 不适用于填充弧。对于填充弧，闭合样式 Arc2D.OPEN 与 Arc2D.CHORD 等效。

下面的语句创建了一个 Arc2D.Float 对象：

```
Arc2D.Float arc = new Arc2D.Float(
    27F, 22F, 42F, 30F, 33F, 90F, Arc2D.PIE);
```

该弧所属椭圆的外切矩形的左上角坐标为(27, 22)，宽度和高度分别为 42 和 30 个像素。弧的起始角度为 33°，并顺时针环绕 90°，闭合方式为扇形。

#### 5. 多边形

在 Java2D 中，多边形是通过从一个顶点移到另一个顶点来创建的。多边形可以由直线、二次曲线和贝塞尔曲线构成。

创建多边形的运动被定义成 GeneralPath 对象，它也位于 java.awt.geom 包中。

创建 GeneralPath 对象时可以不提供任何参数，如下所示：

```
GeneralPath polly = new GeneralPath();
```

GeneralPath 的 moveTo() 方法用于创建多边形的第一个顶点。如果名为 polly 的多边形的第一个顶点坐标为(5, 0)，则可以使用下面的语句：

```
polly.moveTo(5f, 0f);
```

创建第一顶点后，可以使用 lineTo() 方法来创建多边形的边。这个方法接受两个参数：新顶点的 x 和 y 坐标。

下面的语句将 3 条边加入到 polly 对象中：

```
polly.lineTo(205f, 0f);
polly.lineTo(205f, 90f);
polly.lineTo(5f, 90f);
```

方法 lineTo() 和 moveTo() 使用 float 参数来指定坐标点。

要闭合多边形，可以调用方法 closePath()，且不提供任何参数，如下所示：

```
polly.closePath();
```

该方法将当前点与最近的 moveTo() 方法指定的点连接起来，从而将多边形闭合。您也可以不使用这个方法，而使用 lineTo() 方法连接到初始点来闭合多边形。

创建闭合或不闭合的多边形后，可以使用方法 draw() 和 fill() 来绘制它，就像绘制其他图形一样。对象 polly 是一个矩形，它的顶点坐标分别是(5, 0)、(205, 0)、(205, 90)和(5, 90)。

### 13.3.4 绘制对象

定义好渲染属性（如颜色和线宽），并创建要绘制的对象后，便可以绘制 2D 图形。

无论绘制什么对象，都使用相同的 Graphics2D 方法：draw() 用于画边框，fill() 用来填充对象。它们接受一个参数：要绘制的对象。



### 绘制地图

接下来创建一个应用程序，它使用 2D 绘图技术绘制一幅简单的地图。请在编辑器中输入程序清单 13.2 中的代码，并将其保存为文件 Map.java。

程序清单 13.2 完整的 Map.java 源代码

```

1: import java.awt.*;
2: import java.awt.geom.*;
3: import javax.swing.*;
4:
5: public class Map extends JFrame {
6:     public Map() {
7:         super("Map");
8:         setSize(350, 350);
9:         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10:        MapPane map = new MapPane();
11:        add(map);
12:        setVisible(true);
13:    }
14:
15:    public static void main(String[] arguments) {
16:        Map frame = new Map();
17:    }
18:
19: }
20:
21: class MapPane extends JPanel {
22:     public void paintComponent(Graphics comp) {
23:         Graphics2D comp2D = (Graphics2D)comp;
24:         comp2D.setColor(Color.blue);
25:         comp2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
26:             RenderingHints.VALUE_ANTIALIAS_ON);
27:         Rectangle2D.Float background = new Rectangle2D.Float(
28:             0F, 0F, (float)getSize().width, (float)getSize().height);
29:         comp2D.fill(background);
30:         // Draw waves
31:         comp2D.setColor(Color.white);
32:         BasicStroke pen = new BasicStroke(2F,
33:             BasicStroke.CAP_BUTT, BasicStroke.JOIN_ROUND);
34:         comp2D.setStroke(pen);
35:         for (int ax = 0; ax < 340; ax += 10)
36:             for (int ay = 0; ay < 340; ay += 10) {
37:                 Arc2D.Float wave = new Arc2D.Float(ax, ay,
38:                     10, 10, 0, -180, Arc2D.OPEN);
39:                 comp2D.draw(wave);
40:             }
41:         // Draw Florida
42:         GradientPaint gp = new GradientPaint(0F, 0F, Color.green,
43:             350F, 350F, Color.orange, true);
44:         comp2D.setPaint(gp);
45:         GeneralPath fl = new GeneralPath();
46:         fl.moveTo(10F, 12F);
47:         fl.lineTo(234F, 15F);
48:         fl.lineTo(253F, 25F);
49:         fl.lineTo(261F, 71F);
50:         fl.lineTo(344F, 209F);
51:         fl.lineTo(336F, 278F);
52:         fl.lineTo(295F, 310F);
53:         fl.lineTo(259F, 274F);
54:         fl.lineTo(205F, 188F);
55:         fl.lineTo(211F, 171F);
56:         fl.lineTo(195F, 174F);
57:         fl.lineTo(191F, 118F);
58:         fl.lineTo(120F, 56F);
59:         fl.lineTo(94F, 68F);
60:         fl.lineTo(81F, 49F);
61:         fl.lineTo(12F, 37F);
62:         fl.closePath();
63:         comp2D.fill(fl);
64:         // Draw ovals

```



```

65:         comp2D.setColor(Color.black);
66:         BasicStroke pen2 = new BasicStroke();
67:         comp2D.setStroke(pen2);
68:         Ellipse2D.Float e1 = new Ellipse2D.Float(235, 140, 15, 15);
69:         Ellipse2D.Float e2 = new Ellipse2D.Float(225, 130, 15, 15);
70:         Ellipse2D.Float e3 = new Ellipse2D.Float(245, 130, 15, 15);
71:         comp2D.fill(e1);
72:         comp2D.fill(e2);
73:         comp2D.fill(e3);
74:     }
75: }

```

下面是对应用程序 Map 的一些说明。

- 第 2 行导入 java.awt.geom 包中的类。由于第 1 行中的 import java.awt.\* 只导入了 java.awt 中的类，而没有导入其中的包，所以这条导入语句是必不可少的。
- 第 23 行创建了 comp2D 对象，用于执行所有的 2D 绘图操作。这是通过对一个 Graphics 对象进行强制类型转换得到的，该 Graphics 对象表示面板的可视化表面。
- 第 32~34 行创建了一个 BasicStroke 对象，其线宽为 2 个像素。然后使用 Graphics2D 的 setStroke() 方法将其设置为当前画笔。
- 第 35~40 行使用两个嵌套的 for 循环来绘制圆弧，已形成波浪。
- 第 42~43 行创建一种渐变填充图案，其起点和终点坐标分别为(0, 0)和(50,50)，其中起点和终点的颜色分别为绿色和橙黄色。传递给构造函数的最后一个参数为 true，因此填充模式将重复所需的次数，以填满整个对象。
- 第 44 行使用 setPaint() 方法和刚创建的 gp 对象来设置当前的渐变填充图案。
- 第 45~63 行创建一个多边形，并绘制它。该多边形将以绿色到橙黄色的渐变图案填充。
- 第 65 行将当前颜色设置为黑色。这样，接下来的绘制操作中，将使用黑色（而不是渐变图案）进行填充，因为颜色也是填充图案。
- 第 66 行创建一个新的 BasicStroke 对象，且不提供任何参数，这样线宽将为默认值——1 个像素。
- 第 67 行将当前线宽设置为新创建的 BasicStroke 对象 pen2。
- 第 68~70 行创建 3 个椭圆形，它们的外切矩形的左上角坐标分别为(235, 140)、(225, 130)和(245, 130)，高度和宽度都为 15 个像素（因此，实际上为圆）。

图 13.9 显示了该应用程序的运行情况。

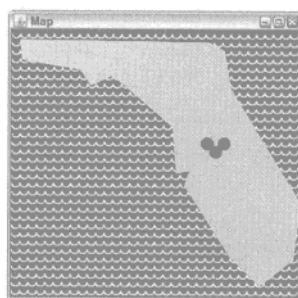


图 13.9 应用程序 Map

## 13.4 总结

现在，您可以使用一些工具来改进 Java 程序的外观。您可以使用 Java2D 在框架、面板和其他用户界面组件上绘制直线、矩形、椭圆、多边形、字体、颜色和图案。

在 Java2D 中，每种绘图操作使用的方法都相同：`draw()` 和 `fill()`。不同的对象是使用 `java.awt.geom` 包中的类来创建的，这些对象被用作 `Graphics2D` 绘图方法的参数。

在下一章中，将介绍如何使用 Java Web Start 技术创建可从网页启动的应用程序。

## 13.5 问与答

问：我对源代码中的大写字母“F”的含义感到迷惑。它被加到坐标的后面，如 `polly.moveTo(5F, 0F)`。为什么在这些坐标中使用的是“F”而不是其他字母？为什么大写字母“F”被用于其他地方？

答：F 和 f 表明一个字面量是浮点数而不是整数，f 和 F 可以互换。如果没有 F 或 f，Java 编译器将认为字面整数是 `int` 值。在 Java 中，很多方法和构造函数接受浮点参数，但能够处理整数，因为整数可被转换为浮点数，而不会改变它的值。因此，诸如 `Arc2D.Float()` 等构造函数可以将 10 和 180 等整数（而不是 10F 和 180F）作为参数。

问：在有关反走样的一节中，提到了名为 `RenderingHint.Key` 的类。为何这个类的名称由两部分组成，它们之间为句点？这意味着什么？

答：使用两部分来标识类表明这是一个内部类。第一部分是包含内部类的类的名称，第二部分是内部类的名称。这个例子表明，内部类 `Key` 位于 `RenderingHint` 类中。

## 13.6 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 13.6.1 问题

- 在 Java 中绘制对象之前，需要什么对象？
  - `Graphics2D`;
  - `WindowListener`;
  - `JFrame`。
- 下列哪个 Java 语句可用来创建 `Color` 对象？
  - `Color c1 = new Color(0F, 0F, 0F);`
  - `Color c2 = new Color(0, 0, 0);`
  - 上述两个语句都可以。
- `getSize().width` 指的是什么？
  - 界面组件的窗口宽度；
  - 框架的窗口宽度；
  - 在 Java 中任何图形用户界面的宽度。

答案

- a。
- c，这两条语句都可用来创建 `Color` 对象。还可以使用十六进制值来创建 `Color` 对象，如下例所示：

```
Color c3 = new Color(0xFF, 0xCC, 0x66);
```

- c，可以对 Java 的任何组件调用 `getSize().width` 和 `getSize().height`。

### 13.6.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

对于下述代码：

```
import java.awt.*;
import javax.swing.*;

public class Result extends JFrame {
    public Result() {
        super("Result");
        JLabel width = new JLabel("This frame is " +
            getSize().width + " pixels wide.");
        add("North", width);
        setSize(220, 120);
    }

    public static void main(String[] arguments) {
        Result r = new Result();
        r.setVisible(true);
    }
}
```

当程序运行时，报告的框架宽度为多少个像素？

- a. 0;
- b. 120;
- c. 220;
- d. 用户显示器的宽度。

答案 a

### 13.7 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 创建一个应用程序，它绘制一个圆，其半径、 $x$  和  $y$  坐标以及颜色都通过命令行参数获得。
2. 创建一个绘制饼图（pie graph）的应用程序。



## 第 14 章

# 开发 Swing 应用程序

很多人都是通过小程序首次接触到 Java 编程语言的，小程序是作为网页一部分运行的安全的小型 Java 程序。

Java Web Start 是一种用于下载并运行程序的协议，它使得可以在 Web 浏览器中像运行小程序那样运行应用程序。

本章介绍如何创建这些从网页启动的 Java 程序，包括如下主题：

- 如何在 Web 浏览器中安装并运行 Java 应用程序？
- 如何发布应用程序文件和运行它？
- Swing 应用程序处理耗时的任务时性能为何会降低？
- 如何使用 SwingWorker 解决这种问题？SwingWorker 是一个在独立线程中完成 Swing 工作的类。

### 14.1 Java Web Start

Java 程序员必须要处理的一个问题是，如何让用户获得 Java 程序。

Java 应用程序需要 Java 解释器，因此要么随应用程序提供 Java 解释器，要么用户的计算机上已经安装了 Java 解释器，否则用户必须自己安装 Java 解释器。对于程序员来说，最容易的解决方案是要求用户从 Sun 公司的网站 (<http://www.java.com>) 下载并安装 Java 运行环境。

不管如何处理 Java 解释器需求，都可以像任何其他程序一样来分发 Java 应用程序，如将它保存在光盘上、放到网站或通过其他方式分发。用户必须运行安装程序来安装它（如果有安装程序）或手工复制文件和文件夹。

小程序更容易分发，因为它们可以由 Web 浏览器运行。但如果是 Java 小程序，用户必须安装带 Java Plug-in 的浏览器。Java Plug-in 作为 Java 运行环境的一部分，同样可从 Sun 公司的网站下载。

相对于应用程序，小程序有几个缺点。最大的缺点是小程序的默认安全策略，它使得小程序无法读写用户计算机上的数据。

Java 使用 Java Web Start 来应对这些软件部署方面的挑战。Java Web Start 是一种运行网页上指出的、存储在 Web 服务器中的 Java 应用程序的方式，其工作原理如下。

1. 程序员使用 JNLP (Java 网络启动协议，是 Java Web Start 的组成部分) 将应用程序及其所需的文件打包成 JAR 文件。
2. 将该文件发布在 Web 服务器上，并在网页上使用一个链接指向该文件。
3. 用户使用浏览器加载该页面，并单击该链接。
4. 如果用户没有 Java 运行环境，则打开一个对话框，询问是否下载并安装 Java 运行环境。当前，完全安装的大小为超过 65M，通过 56K 的 Internet 连接下载需要 30~45 分钟，如果使用高速连接，则需要 3~5 分钟。

5. Java 运行环境安装并运行该程序，打开新窗口或其他界面组件，就像其他应用程序一样。程序被保存在缓存中，以后再次运行它时不必重新安装。

要查看其实际情况，可访问 Sun 公司的 Java Web Start 网站 <http://java.sun.com/products/javawebstart>，并单击链接 Code Samples & Apps，再单击链接 Demos。这个 Web Start Demos 页面包含几个 Java 应用程序的图片，每个图片都有一个 Launch 按钮，单击它可运行相应的 Java 应用程序，如图 14.1 所示。

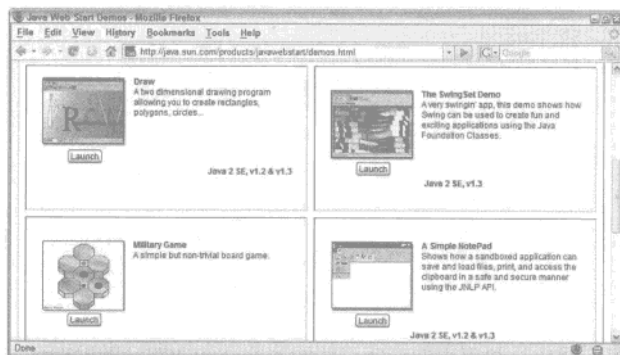


图 14.1 在网页上链接 Web Start 应用程序

单击某个应用程序的 Launch 按钮。如果还没有安装 Java 运行环境，将打开一个对话框，询问是否要下载并安装它。

Java 运行环境中包括 Java Plug-in，它是一个 Java 解释器，可为浏览器（如 Internet Explorer 和 Mozilla）添加对当前 Java 版本的支持。Java 运行环境还可用于运行 Java 应用程序，无论这些程序是否使用了 Java Web Start。

使用 Java Web Start 运行应用程序时，一个标题屏幕将在计算机上显示片刻，然后出现应用程序的图形用户界面。

#### 注意

如果安装了 JDK，计算机很可能已安装了 Java 运行环境。

图 14.2 是 Sun 公司提供的演示应用程序，它是一个军事策略游戏，其中 3 个黑点试图防止 1 个红点进入它们的势力范围。

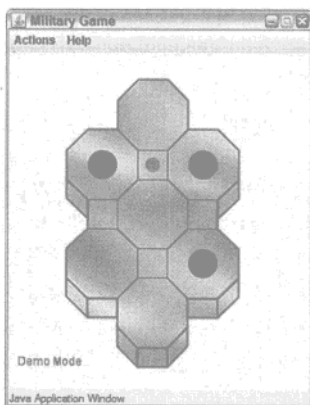


图 14.2 运行 Java Web Start 应用程序

如图 14.2 所示, 该应用程序看起来与其他 Java 应用程序没有任何不同。与 (在网页中运行的) 小程序不同的是, Java 应用程序通过 Java Web Start 启动, 并在自己的窗口中运行, 与从命令行运行它们的效果相同。

Java Web Start 应用程序的不同之处在于它为用户提供的安全性。当应用程序试图执行诸如读写文件等操作时, 将询问用户是否允许。

例如, 另一个演示程序是一个文本编辑器。当用户首次试图保存文件时, 将打开如图 14.3 所示的 Security Warning 对话框。

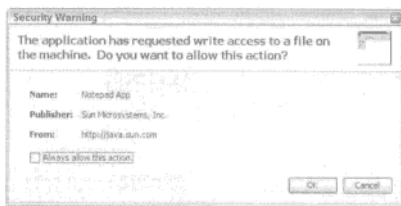


图 14.3 选择应用程序的安全权限

如果用户不允许某些操作, 则应用程序将不能发挥其全部功能。触发安全对话框的操作是默认情况下小程序不允许的操作: 读写文件、加载网络资源等。

应用程序通过 Java Web Start 运行后, 将被存储在缓存中, 这样以后再次运行它时无需重新安装。唯一例外的情况是存在应用程序的新版本。在这种情况下, 将自动下载新版本并将其安装到原来的位置。

可直接运行 Java Web Start 应用程序查看器来查看被缓存的应用程序、运行它们和修改其设置。这个应用程序名为 javaws.exe, 可在 java 以及 JDK 中的其他命令行程序所在的文件夹中找到。另外, 还应该有一个针对 Java Web Start 的菜单项, 这个菜单项是在安装期间加入的。

#### 注意

尽管第一次是使用 Web 浏览器来运行 Java Web Start 应用程序, 但并非一定要这样做。为明白这一点, 可运行 Java Web Start 查看器, 然后选择一个程序, 并选择菜单 Application/Install Shortcuts。这样, 一个运行该程序的快捷方式将被加入到桌面上, 您可以通过它来运行该程序, 而无需使用浏览器。

如果 Java Web Start 应用程序被存储在一个包含数字签名的 Java 存档文件中, 则可以覆盖其默认的安全限制。应用程序运行时, 将向用户显示带签名的安全认证, 其中列出了程序的作者以及认证机构, 并询问用户接受还是拒绝。除非用户接受, 否则应用程序将不会运行。

## 14.2 使用 Java Web Start

只要提供 Java 应用程序的 Web 服务器被配置成能够使用 Java Web Start 技术, 且类文件及其所需的其他文件被打包在一起, 就可使用 Java Web Start 来运行它们。

为使 Java 应用程序能够利用 Java Web Start, 必须将其文件保存到 JAR 文件中, 为其创建一个特殊的 Java Web Start 配置文件, 并将这两个文件上传到 Web 服务器。

配置文件必须使用 Java 网络启动协议 (Java Network Launching Protocol, JNLP) 来创建。该协议采用可扩展的标记语言 (XML) 文件格式指定 Java 应用程序的主类文件、JAR 文件及其他信息。

#### 注意

XML 将在第 20 章介绍。由于 JNLP 文件的格式相对直观, 因此不需要了解太多有关 XML 文件的知识就可创建 JNLP 文件。

下面将使用 Java Web Start 加载并运行 PageData 应用程序，该程序显示有关网页的信息。作为准备工作，先将该项目的类文件复制到您用作 Java 编程工作空间的文件夹中。

### 14.2.1 创建 JNLP 文件

首先需要将应用程序的类文件以及它所需要的其他任何文件打包成 Java 存档 (JAR) 文件。如果使用的是 JDK，可通过如下命令创建 JAR 文件：

```
jar -cf PageData.jar PageData.class
```

该命令将创建一个名为 PageData.jar 的 JAR 文件，其中包含类文件。

接下来为该应用程序创建一个图标图形，在成功加载该程序后将显示该图形，它还将被用作菜单和桌面上的图标。图标可以是 GIF 或 JPEG 格式，但其大小应为 64 像素宽和 64 像素高。

就这里而言，如果您不想创建新图标，可使用本书配套资源中的 pagedataicon.gif 文件（从人民邮电出版社网站 [www.ptpress.com.cn](http://www.ptpress.com.cn) 下载）。

最后需要做的是创建用于描述该应用程序的 JNLP 文件。程序清单 14.1 包含了用于描述 PageData 应用程序的 JNLP 文件。打开字处理器程序，并输入该程序清单中的代码，然后将其保存为 PageData.jnlp。

程序清单 14.1 PageData.jnlp 的完整源代码

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <!-- JNLP File for the PageData Application -->
3: <jnlp
4:   codebase="http://www.cadenhead.org/book/java-21-days/java"
5:   href="PageData.jnlp">
6:   <information>
7:     <title>PageData Application</title>
8:     <vendor>Rogers Cadenhead</vendor>
9:     <homepage href="http://www.java21days.com"/>
10:    <icon href="pagedataicon.gif"/>
11:    <offline-allowed/>
12:  </information>
13:  <resources>
14:    <j2se version="1.6"/>
15:    <jar href="PageData.jar"/>
16:  </resources>
17:  <security>
18:    <all-permissions/>
19:  </security>
20:  <application-desc main-class="PageData"/>
21: </jnlp>
```

JNLP 文件的结构与将 Java 小程序放到网页上所需的 HTML 标记类似。位于“<”和“>”之间的内容是标记，标记之间的内容是标记描述的信息。在信息之前有一个起始标记，而信息之后有一个结束标记。

例如，程序清单 14.1 中第 7 行包含下列文本：

```
<title>PageData Application</title>
```

按从左到右的次序，该行代码包含起始标记<title>、文本 PageData Application 以及结束标记</title>。位于标记之间的文本——PageData Application——是该应用程序的标题。加载该应用程序时 Java Web Start 将显示该标题，该标题还将用于菜单和快捷方式中。

起始标记与结束标记之间的区别是结束标记以斜杠 (/) 开头，而起始标记则没有。在第 8 行中，<vendor>是起始标记，</vendor>是结束标记，而这两个标记之间的内容是创建该应用程序的厂商名称。这里使用了作者的姓名。您也可删除它，并替换为自己的姓名，但要注意不要修改两边的<vendor>和



</vendor>标记。

有些标记只有起始标记，如第 11 行：

```
<offline-allowed/>
```

这个 offline-allowed 标记指出，即使用户没有连接到因特网也可运行该应用程序。如果在 JNLP 文件中省略它，则正好相反，即用户必须连接到因特网才能运行该程序。

在 XML 中，所有不需要结束标记的标记都用“/”（而不是“>”）结尾。

标记还可以有属性，它是在 XML 文件中定义信息的另一种方法。属性就是标记内的一个名称，后跟“=”以及用引号括起的文本。

例如，程序清单 14.1 中的第 9 行内容为：

```
<homepage href="http://www.java21days.com"/>
```

这是一个 homepage 标记，有一个 href 属性。引号中的文本用于将该属性的值设置为 http://www.java21days.com。它定义了该应用程序的主页——用户如果需要了解关于该程序及其如何工作的更多信息，则应该访问该主页。

JNLP 文件 PageData 定义了一个简单的 Java Web Start 应用程序，后者运行时没有任何安全限制，这是在第 17~19 行定义的：

```
<security>
  <all-permissions/>
</security>
```

除上面所介绍的标记外，程序清单 14.1 还定义了 Java Web Start 所需要的其他信息。

第 1 行指出该文件使用 XML 和 UTF-8 字符集。这一行也可用于其他任何应用程序的 JNLP 文件中。

第 2 行是注释。与 Java 中的其他注释相同，其唯一用途是使文件更易于阅读。Java Web Start 将忽略它。

第 3~21 行是 jnlp 元素，它必须将其他所有配置 Web Start 的标记括起。

该标记有两个属性：codebase 和 href，用于指示在哪里可获得该应用程序的 JNLP 文件。属性 codebase 是 JNLP 文件所在文件夹的统一资源定位符（URL）；属性 href 是该文件的名称或包含一个文件夹和名称的相对 URL（如 pub/PageData.jnlp）。

在程序清单 14.1 中，该属性指出应用程序的 JNLP 文件位于如下网络地址：

```
http://www.cadenhead.org/book/java-21-days/java/PageData.jnlp
```

第 6~12 行中的 information 元素定义关于应用程序的信息。该元素可包含其他 XML 元素，在程序清单 14.1 中，information 元素包含标记 title、vendor、homepage、icon 和 offline-allowed。

其中 title、vendor、homepage 和 offline-allowed 元素在前面已介绍过。

第 10 行中的 icon 元素包含一个 href 属性，它指出了程序图标的名称（或文件夹位置和名称）。与 JNLP 文件中所有文件引用一样，该元素使用 codebase 属性确定资源的完整 URL。在本例中，icon 元素的 href 属性是 pagedataicon.gif，codebase 是 http://www.cadenhead.org/book/java21days/java，因此该图标文件位于如下网络地址：

```
http://www.cadenhead.org/book/java21days/java/pagedataicon.gif
```

第 13~16 行中的 resources 元素定义了应用程序运行时使用的资源。

j2se 元素有一个 version 属性，它指出了用于运行该应用程序的 Java 解释器的版本。该属性可指定通用版本（如 1.5 或 1.6）、特定版本（如 1.6-beta）或多个版本（在通用版本号后跟一个加号）。例如，标记 <j2se version="1.4+"> 指出该应用程序应该使用版本 1.4 以上的 Java 解释器来运行。

#### 注意

使用 j2se 元素指定多个 Java 版本时，Java Web Start 将不使用 beta 版本来运行应用程序。使用 beta 版本来运行应用程序的唯一方式是明确指定使用该版本。

jar 元素有一个 href 属性, 指定应用程序的 JAR 文件。该属性可以是一个文件名, 或是到一个文件夹和文件名的引用, 它使用 codebase。在 PageData 的示例中, JAR 文件的位置是 <http://www.cadenhead.org/book/java21days/java/PageData.jar>。

“application-desc” 元素指出应用程序的主类文件及执行该类时应使用的参数。

“main-class” 属性标识类文件的名称, 指定时不需要使用文件扩展名.class。

如果运行该类时需要使用一个或多个参数, 则应在起始标记 <application-desc> 和结束标记 </application-desc> 之间加入 argument 元素。

下面的 XML 代码指定运行 PageData 类时需要使用两个参数: <http://java.sun.com> 和 yes:

```
<application-desc main-class="PageData">
  <argument>http://java.sun.com</argument>
  <argument>yes</argument>
</application-desc>
```

创建 PageData.jnlp 文件后, 修改程序清单 14.1 中的第 5 行, 使之与您的 Web 服务器上用于存放该应用程序的 JAR 文件、图标文件和 JNLP 文件的文件夹路径一致。

将这个项目的 3 个文件上传到该文件夹, 然后运行浏览器, 使用 JNLP 文件的完整地址加载 JNLP 文件。如果 Web 服务器被配置成支持 Java Web Start, 该应用程序将被加载并开始运行, 如图 14.4 所示。

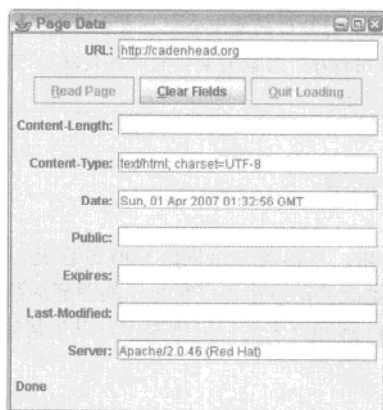


图 14.4 使用 Java Web Start 运行 PageData

要让这个程序能够不受限制地运行, 必须对文件 PageData.jar 采用数字签名, 这需要认证授权机构 (如 Thawte 或 Verisign) 提供的服务。

出于测试的目的, 可使用 JDK 中的工具 keytool 和 jarsigner 来生成一个密钥 (key), 并使用它对 JAR 文件进行数字签名。

首先, 需要使用 keytool 来生成一个密钥, 并给它指定别名和密码:

```
keytool -genkey -alias examplekey -keypass swordfish
```

参数 -genkey 生成新密钥, 在这个例子中名为 examplekey, 其密码为 swordfish。如果 keytool 是首次被使用, 将提示您提供一个密码, 用于控制对密钥数据库的访问, 这个密钥数据库被称为 keystore。

将密钥放入密钥库中后, 便可以通过 jarsigner 来使用它对存档文件进行签名。这个工具要求提供密钥库、密钥密码和密钥别名。下面的例子演示了如何使用密钥 examplekey 对存档文件 PageData.jar 进行数字签名:

```
jarsigner -storepass secret -keypass swordfish PageData.jar examplekey
```

在这个例子中, 密钥库密码为 secret。用于对存档文件进行签名的安全证明的有效期为 90 天, Java Web Start 应用程序运行时, 它被描述为“来源不可信 (untrusted source)”。

**注意**

没有避免被描述为“不可信”的简单途径。建立信任的唯一方式是通过专业证书授权机构的认证。

## 14.2.2 在服务器上支持 Web Start

如果服务器不支持 Java Web Start (多数情况下都不支持,因为它是一种相对较新的技术),将可能看到 JNLP 文件的文本内容被加载到页面,但应用程序不会打开。

必须对 Web 服务器进行配置,才能使之知道 JNLP 文件是一种新的数据类型,能导致 Java 应用程序运行。这通常需要将文件扩展名 JNLP 与 MIME 类型相关联。

MIME 是 Multipurpose Internet Mail Extensions (多用途因特网邮件扩展)的缩写,它是一种协议,用于定义 Internet 内容,如电子邮件地址、邮件附件及其他任何可由 Web 服务器发送的文件。

在 Apache Web 服务器上,服务器管理员通过添加下列代码到服务器的 mime.types (或 mime.types) 文件中,即可支持 JNLP:

```
application/x-java-jnlp-file JNLP
```

**警告**

用 Java Web Start 运行的应用程序与通过其他方式运行的应用程序应该完全相同。但是,在图形用户界面上,对组件的空间分配上存在问题。在 Windows 系统上,为了通过 Java Web Start 启动应用程序,可能需要将它的高度增加 50 像素。否则,其文本框高度将导致数字无法正常显示。

## 14.2.3 其他 JNLP 元素

JNLP 文件中还有其他一些元素,可影响 Java Web Start 的性能。

例如,可使用 JNLP 元素来更改启动应用程序时显示的图形标题、运行具有不同安全权限的已签名应用程序、使用不同版本的 Java 解释器来运行应用程序,以及其他一些选项。

### 1. Security

默认情况下,所有 Java Web Start 应用程序将不能访问用户计算机上的某些功能,除非用户赋予该权限。这与对 Java 小程序的安全限制功能类似。

如果应用程序的 JAR 文件已经采用数字签名来验证其可靠性,则可使用 security 元素,使它在运行时没有安全限制。

该元素将放入 jnlp 元素中,它自己包含一个元素: all-permissions。要解除对应用程序的安全限制,可添加如下代码到相应的 JNLP 文件中:

```
<security>
  <all-permissions/>
</security>
```

### 2. Descriptions

如果要通过 Java Web Start 为用户提供关于应用程序的更多信息,则可在 information 元素内部使用一个或多个 description 元素。

通过 description 元素的 kind 属性,可提供 4 种描述。

- kind = "one-line": 简洁的单行描述,用于 Web Start 应用程序列表中。
- kind = "short": 一段描述,当有足够空间时将显示该描述。

- `kind = "tooltip"`: 一个工具提示描述。
- 无 `kind` 属性: 默认描述, 当没有指定其他描述时将使用它。

这 4 种描述都是可选的。例如, 下列内容将为 PageData 应用程序提供描述:

```
<description>The PageData application.</description>
<description kind="one-line">An application to learn more about web
servers and pages.</description>
<description kind="tooltip">Learn about web servers and
pages.</description>
<description kind="short">PageData, a simple Java application that
takes a URL and displays information about the URL and the web
server that delivered it.</description>
```

### 3. Icons

PageData 的 JNLP 文件包括一个大小为  $64 \times 64$  像素的图标 `pagedataicon.gif`, 它有两个用途。

- 当 Java Web Start 加载 PageData 应用程序时, 将在该程序名称和作者旁边的一个窗口中显示该图标。

- 如果将 PageData 图标添加到用户桌面上, 则将使用另一个尺寸 ( $32 \times 32$  像素) 来显示该图标。

在应用程序加载过程中, 可使用另一个 `icon` 元素指定将显示在图标、标题和作者位置的图形。该图形叫做应用程序的启动画面 (splash screen), 它通过使用 `kind = "splash"` 属性来指定, 如下例中所示:

```
<icon kind="splash" href="pagedatasplash.gif" width="300" height="200">
```

其中的 `width` 和 `height` 属性 (也可用于其他类型的图标图形) 以像素为单位, 指定图像的显示尺寸。

这里第 2 个 `icon` 元素也应该放在 `information` 元素之中。

## 14.3 使用 SwingWorker 改善性能

Swing 应用程序的响应速度在很大程度上取决于它在处理耗时任务时如何响应用户输入。

应用程序通常在一个线程中执行任务, 因此如果某项任务 (如加载大型文件或分析 XML 文档的数据) 需要很长时间才能完成, 用户可能在此期间感觉到性能降低。

Swing 程序还要求所有用户界面组件都在同一个线程中运行。

为满足这种需求, 同时提高性能, 最佳的方式是使用 `SwingWorker`, 这是 `javax.swing` 包中新增的一个类, 用于在独立的工作线程中运行耗时的任务并报告结果。

`SwingWorker` 是一个抽象类, 需要工作线程的应用程序必须继承它, 如下所示:

```
public class DiceWorker extends SwingWorker {
    // ...
}
```

在新类中, 应覆盖方法 `doInBackground()`, 如下面的示例所示。该示例掷六面骰子非常多次, 并跟踪结果:

```
doInBackground() {
    for (int i = 0; i < timesToRoll; i++) {
        int sum = 0;
        for (int j = 0; j < 3; j++) {
            sum += Math.floor(Math.random() * 6);
        }
        result[sum] = result[sum] + 1;
    }
    return result;
}
```

接下来的项目是一个 Swing 应用程序, 它按用户指定的次数掷六面骰子, 并以表格方式显示结果。16 个文本框表示可能的值——3~18。

这个应用程序包含两个类: 框架 `DiceRoller` 和 `SwingWorker` 对象 `DiceWorker`, 前者用于放置图形

用户界面，后者处理掷骰子的工作。

该应用程序允许用户掷骰子数千甚至数百万次，因此将这项任务放在一个工作线程中，以确保 Swing 界面能够快速响应用户输入。

程序清单 14.2 是 DiceWorker 类的代码。

程序清单 14.2 DiceWorker.java 的完整源代码

---

```

1: import javax.swing.*;
2:
3: public class DiceWorker extends SwingWorker {
4:     int timesToRoll;
5:
6:     // set up the Swing worker
7:     public DiceWorker(int timesToRoll) {
8:         super();
9:         this.timesToRoll = timesToRoll;
10:    }
11:
12:    // define the task the worker performs
13:    protected int[] doInBackground() {
14:        int[] result = new int[16];
15:        for (int i = 0; i < this.timesToRoll; i++) {
16:            int sum = 0;
17:            for (int j = 0; j < 3; j++) {
18:                sum += Math.floor(Math.random() * 6);
19:            }
20:            result[sum] = result[sum] + 1;
21:        }
22:        // transmit the result
23:        return result;
24:    }
25: }

```

---

您可以编译这个类，但在创建下一个类 DiceRoler 前，将无法使用它做任何事情。

Swing Worker 只需要一个方法——在后台执行任务的 `doInBackground()`。该方法的访问控制级别必须为 `protected`，它返回结果。DiceWorker 创建了一个包含 16 个元素的数组，用户存储掷骰子的结果。

其他类使用该 Swing Worker 时需要按 3 步进行。

1. 调用其构造函数 `DiceWorker(int)`，并将参数设置为掷骰子的次数。
2. 调用其 `addChangeListener(Object)` 方法添加一个监听器，当任务结束时将通知该监听器。
3. 调用其 `execute()` 方法开始执行任务。

`execute()` 方法导致调用 Worker 的 `doInBackground()` 方法。

属性更改监听器是一个来自 `java.bean` 的事件监听器。`java.bean` 是 `JavaBean` 包，指定了用户界面中的组件如何交互。

在这个例子中，一个 Swing worker 需要宣告其工作已完成，这可能是该 worker 开始工作后很久的事情。监听器是处理这种通知的最佳方式，因为它们让图形用户界面无需处理其他工作。

属性更改监听器接口有一个方法：

```

public void propertyChange(PropertyChangeEvent event) {
    // ...
}

```

程序清单 14.3 所示的 DiceRoller 类提供了用户界面，可显示掷骰子的结果以及让用户重新开始。

程序清单 14.3 完整的 DiceRoller.java 源代码

---

```

1: import java.awt.*;
2: import java.awt.event.*;
3: import java.beans.*;
4: import javax.swing.*;
5:
6: public class DiceRoller extends JFrame implements ActionListener,

```

---

```

7: PropertyChangeListener {
8:
9:     // the table for dice-roll results
10:    JTextField[] total = new JTextField[16];
11:    // the "Roll" button
12:    JButton roll;
13:    // the number of times to roll
14:    JTextField quantity;
15:    // the Swing worker
16:    DiceWorker worker;
17:
18:    public DiceRoller() {
19:        super("Dice Roller");
20:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21:        setSize(800, 125);
22:
23:        // set up top row
24:        JPanel topPane = new JPanel();
25:        GridLayout paneGrid = new GridLayout(1, 16);
26:        topPane.setLayout(paneGrid);
27:        for (int i = 0; i < 16; i++) {
28:            // create a textfield and label
29:            total[i] = new JTextField("0", 4);
30:            JLabel label = new JLabel((i + 3) + ": ");
31:            // create this cell in the grid
32:            JPanel cell = new JPanel();
33:            cell.add(label);
34:            cell.add(total[i]);
35:            // add the cell to the top row
36:            topPane.add(cell);
37:        }
38:
39:        // set up bottom row
40:        JPanel bottomPane = new JPanel();
41:        JLabel quantityLabel = new JLabel("Times to Roll: ");
42:        quantity = new JTextField("0", 5);
43:        roll = new JButton("Roll");
44:        roll.addActionListener(this);
45:        bottomPane.add(quantityLabel);
46:        bottomPane.add(quantity);
47:        bottomPane.add(roll);
48:
49:        // set up frame
50:        GridLayout frameGrid = new GridLayout(2, 1);
51:        setLayout(frameGrid);
52:        add(topPane);
53:        add(bottomPane);
54:
55:        setVisible(true);
56:    }
57:
58:    // respond when the "Roll" button is clicked
59:    public void actionPerformed(ActionEvent event) {
60:        int timesToRoll = 0;
61:        try {
62:            // turn off the button
63:            timesToRoll = Integer.parseInt(quantity.getText());
64:            roll.setEnabled(false);
65:            // set up the worker that will roll the dice
66:            worker = new DiceWorker(timesToRoll);
67:            // add a listener that monitors the worker
68:            worker.addPropertyChangeListener(this);
69:            // start the worker
70:            worker.execute();
71:        } catch (Exception exc) {
72:            System.out.println(exc.getMessage());
73:            exc.printStackTrace();
74:        }
75:    }
76:
77:    // respond when the worker's task is complete
78:    public void propertyChange(PropertyChangeEvent event) {

```

```

79:         try {
80:             // get the worker's dice-roll results
81:             int[] result = (int[]) worker.get();
82:             // store the results in text fields
83:             for (int i = 0; i < result.length; i++) {
84:                 total[i].setText("" + result[i]);
85:             }
86:         } catch (Exception exc) {
87:             System.out.println(exc.getMessage());
88:             exc.printStackTrace();
89:         }
90:     }
91:
92:     public static void main(String[] arguments) {
93:         new DiceRoller();
94:     }
95: }

```

DiceRoller 的大部分代码都用于创建并排列用户界面组件：16 个文本框、一个名为 Times to Roll 的文本框和一个 Roll 按钮。

方法 actionPerformed() 响应用户单击 Roll 按钮，它创建负责掷骰子的 Swing Worker、添加一个属性变更监听器以及开始工作。

在第 70 行调用了 worker.execute() 方法，这导致 Worker 的 doInBackground() 方法被调用。

当 Worker 完成掷骰子的工作后，DiceRoller 的 propertyChange() 方法将收到一个属性变更事件。

该方法通过调用 Worker 的 get() 方法来获取 doInBackground() 的结果（第 81 行），而该结果必须强制转换为整型数组：

```
int[] result = (int[]) worker.get();
```

图 14.5 说明了该应用程序的运行情况。

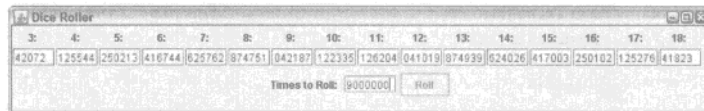


图 14.5 以表格方式显示 SwingWorker 生成的掷骰子结果

## 14.4 总结

Java 有很多功能使其适合用于开发应用程序，本章讨论了其中的两种：基于浏览器的程序部署以及通过使用线程改善 Swing 程序的性能。Java Web Start 模糊了 Java 应用程序和小程序之间的区别。

通过使用 Web Start，用户不再需要运行一个安装程序来安装 Java 应用程序和执行 Java 类的解释器。Web Start 将自动完成这些事情，只要用户的浏览器已经配置成使用 Java 2 运行环境即可。

对 Web Start 的支持是通过 Java 网络启动协议（Java Network Launching Protocol, JNLP）提供的，它采用 XML 文件格式来定义和设置 Java Web Start。

SwingWorker 类将耗时任务放在独立的线程中运行，从而改善了 Swing 应用程序的性能。为启动和终止线程所需做的所有工作都有这个类在幕后处理。

创建 SwingWorker 的子类时，可将重点放在必须执行的任务上。

## 14.5 问与答

问：我编写了一个小程序，并想使用 Java Web Start 来部署它。我是否应该将它转换成一个应用程序？还是可不作修改直接运行它？

答:如果将小程序转换成应用程序仅仅是为了使用 Web Start 来运行它,则可能没有必要。`applet-desc` 标记使得可以在 Java Web Start 中运行小程序而不做任何修改。需要进行这种转换的唯一原因是您想更改程序的其他方面,例如要将 `init()` 方法更换为构造函数。

问:如何确定 `SwingWorker` 对象是否完成了工作?

答:调用其 `isDone()` 方法。如果任务已执行完毕,该方法将返回 `true`。

需要注意的是,无论任务是如何完成的,该方法都将返回 `true`,因此如果任务取消、中断或失败,它也将返回 `true`。

方法 `isCanceled()` 可用于检查任务是否被取消。

## 14.6 小测验

请回答下述 3 个问题,以复习本章介绍的内容。

### 14.6.1 问题

1. 必须实现哪个接口,以便 `SwingWorker` 执行完任务时通知它?
  - a. `ActionListener`;
  - b. `PropertyChangeListener`;
  - c. `SwingListener`。
2. 哪个 XML 元素用于标识有关 Java Web Start 应用程序的名称、作者和其他信息?
  - a. `jnl`;
  - b. `information`;
  - c. `resources`。
3. Java Web Start 应用程序有哪些安全限制?
  - a. 与小程序相同;
  - b. 与应用程序相同;
  - c. 由用户选择。

答案

1. b, 任务完成后, `java.beans` 包中的 `PropertyChangeListener` 将收到 `propertyChange()` 事件。
2. b, 包含在起始标记 `<information>` 和结束标记 `</information>` 之间的元素描述了应用程序。
3. c, 相对于小程序, Java Web Start 应用程序的限制很少,但它们不能执行一些重要功能,如存储文件和打开网络连接。如果应用程序运行时用户显式地授予这些权限,将不存在这些限制。

### 14.6.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题,请回答该问题,而不要查看本章的内容,也不要使用 Java 编译器对代码进行测试。

给定如下代码:

```
import java.awt.*;
import javax.swing.*;

public class SliderFrame extends JFrame {
    public SliderFrame() {
```



```
        super();  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        Container pane = getContentPane();  
        JSlider value = new JSlider(0, 255, 100);  
        getContentPane().add(value);  
        setSize(325, 150);  
        setVisible(true);  
    }  
  
    public static void main(String[] arguments) {  
        new SliderFrame();  
    }  
}
```

试图编译并运行上述源代码时，将发生什么事情？

- a. 它将正确编译并运行，不产生任何错误。
- b. 它将正确编译，但不在窗口中显示任何内容。
- c. 它不能正常编译，因为内容面板为空。
- d. 由于 new SlideFrame()语句，它将不能正确编译。

答案 b

## 14.7 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 将本书前两周课程中创建的应用程序之一转换为可使用 Java Web Start 启动的程序。
2. 创建一个新的 JNLP 文件，它使用 1.3 版 Java 解释器运行 PageData 应用程序，并要求用户运行该程序时必须连接到 Internet。





## 第 3 周课程

### Java 编程

第 15 章 输入和输出

第 16 章 序列化和查看对象

第 17 章 通过 Internet 进行通信

第 18 章 使用 JDBC 访问数据库

第 19 章 读写 RSS Feed

第 20 章 XML Web 服务

第 21 章 编写 Java Servlets 和 Java Server Page

## 第 15 章

# 输入和输出

您使用 Java 创建的很多程序都需要同某种数据源进行交互。将信息存储在计算机中的方式很多，包括硬盘或光盘中的文件、网站上的网页，甚至计算机内存。

您可能猜想，对于每一种不同的存储设备，处理的方式也不同。幸运的是，情况并非如此。

在 Java 中，可以使用被称作流的通信系统来存储和检索信息，流是在 `java.io` 包中实现的。

本章将介绍如何创建输入流和输出流以读取和存储信息。

- 字节流：用于处理字节、整数和其他简单数据类型。
- 字符流：用于处理文本文件和其他文本源。

知道如何处理输入流后，便可以以同样的方式处理所有的数据，不管是它来自磁盘、Internet 还是另一个程序。对于使用输出流来传输数据，情况也是如此。

### 注意

`java.nio` 包提供了其他输入/输出编程技术，这个包主要用于网络编程，因此将在第 17 章介绍。

## 15.1 流

在 Java 中，所有数据都是使用流读写的。流就像水流一样，将数据从一个地方带到另一个地方。

流是程序中的数据所经历的路径。输入流将数据从数据源传递给程序，而输出流将数据从程序发送到某个目的地。

本章介绍两种流：字节流和字符流。字节流传送 0~255 的整数。很多类型的数据都可以表示为字节格式，包括数字数据、可执行程序、Internet 通信和字节码（Java 虚拟机运行的类文件）。

实际上，每种数据都可以表示为单个字节或一系列的字节。

字符流是一种特殊的字节流，只能处理文本数据。它不同于字节流，因为 Java 字符集支持 Unicode——该标准包含很多无法使用字节来表示的字符。

对于任何涉及文本的数据，都应使用字符流，这包括文本文件、网页以及其他常见的文本类型。

### 15.1.1 使用流

在 Java 中，使用字节流和字符流的步骤基本相同。使用特定的 `java.io` 类之前，有必要介绍一下创建和使用流的步骤。

对于输入流，第一步是创建一个与数据源相关联的对象。例如，如果数据源是硬盘上的文件，可以将一个 `FileInputStream` 对象与之关联起来。

有了流对象后，可以使用该对象的方法来从流中读取信息。`FileInputStream` 有一个 `read()` 方法，它从文件中读取字节。

从流中读取完信息后，调用方法 `close()` 来指出已完成对流的使用。

对于输出流，首先要创建一个与数据目的地相关联的对象。这样的对象是从类 `BufferedWriter` 派生而来的，后者是一种创建文本文件的有效方式。

要将信息发送给输出流的目的地，最简单的方式是使用方法 `write()`。例如，`BufferWriter` 的 `write()` 方法将单个字符发送给输出流。

和输入流一样，没有其他信息需要发送时，应调用输出流的 `close()` 方法。

### 15.1.2 过滤流

要使用流，最简单的方式是，创建它，然后调用其方法来发送或接收数据——这取决于它是输出流还是输入流。

对于本章将介绍的很多类，如果在读写数据之前将过滤器与流关联起来，都将获得更精致的结果。

过滤器是一种流，它修改了现有流的处理方式。来看拦河水坝，它控制从上游到下游的流量。水坝就是一种过滤器——如果没有它，将无法控制流量。

对流使用过滤器的步骤如下：

1. 创建一个与数据源或数据目的地相关联的流；
2. 将一个过滤器与流关联起来；
3. 通过过滤器（而不是流）来读写数据。

对于过滤器，可调用的方法与流相同：有 `read()` 和 `write()` 方法，就像没有被过滤的流一样。

甚至可以将过滤器与另一个过滤器关联起来，从而实现这样的信息路径：输入流与文本文件相关联，它被一个西班牙语到英语的翻译过滤器过滤，而后者被一个脏字过滤器过滤，信息最后被发送到目的地——要阅读它的人。

如果这过于抽象而难以理解，接下来的几节将介绍如何使用。

### 15.1.3 处理异常

`java.io` 包中有一些异常，这些异常在您使用文件和流时可能发生。

当您试图使用一个不存在的文件来创建流或文件对象时，将发生 `FileNotFoundException` 异常。

通过输入流来读取文件时，如果过早地到达文件尾，将发生 `EOFException` 异常。

这些异常都是 `IOException` 的子类。为处理所有这些异常，一种方式是将所有的输入和输出流放在一个捕获 `IOException` 异常的 `try-catch` 块中。然后，在 `catch` 块中调用异常的 `toString()` 或 `getMessage()` 方法，以了解有关异常的更详细的信息。

## 15.2 字节流

字节流要么是 `InputStream` 的子类，要么是 `OutputStream` 的子类。这些类都是抽象类，因此不能通过直接创建这些类的对象来创建字节流，而必须通过它们的子类来创建流，如下所示。

- `FileInputStream` 和 `FileOutputStream`：用于磁盘、光盘或其他存储设备中的文件的字节流。
  - `DataInputStream` 和 `DataOutputStream`：被过滤的字节流，从中可读取诸如整数和浮点数等数据。
- `InputStream` 是所有输入流的超类。

## 文件流

您使用的字节流通常是文件流，它用于同磁盘、光盘或其他存储设备中的文件交换数据，这些文件可以使用文件夹路径和文件名来引用。

您可以将字节发送给文件输出流，也可以从文件输入流中读取字节。

### 1. 文件输入流

文件输入流可使用构造函数 `FileInputStream(String)` 来创建，其中的 `String` 参数是文件名。您可以在文件名中指定路径，这样文件可以不位于装载它的类所在的文件夹中。下面的语句使用文件 `scores.dat` 创建了一个文件输入流：

```
FileInputStream fis = new FileInputStream("scores.dat");
指出路径的方式随平台而异，下面的例子在 Windows 系统上读取文件：
FileInputStream f1 = new FileInputStream("\\data\\calendar.txt");
```

#### 注意

由于 Java 在转义码中使用反斜杠，因此在 Windows 上指定路径时，必须使用 `\\` 替代 `\`。

下面是一个在 Linux 系统上读取文件的例子：

```
FileInputStream f2 = new FileInputStream("/data/calendar.txt");
一种更好的引用路径的方式是，使用 File 类中的类变量 separator，这种方法适用于任何操作系统：
char sep = File.separator;
FileInputStream f2 = new FileInputStream(sep + "data"
    + sep + "calendar.txt");
```

创建文件输入流后，可以调用 `read()` 方法来读取流中的字节。该方法返回一个整数，它是流中的下一个字节。如果返回 -1（这不是字节值），则表明已到达文件流的末尾。

要从流中读取多个字节，可调用其 `read(byte[], int, int)` 方法。该方法的参数如下：

1. 一个用于存储数据的字节数组；
2. 数组的第一个元素，应存储数据的第一个字节；
3. 要读取的字节数。

与其他 `read()` 方法不同，该方法并不返回流中的数据，而是返回一个整数，表示读取的字节数。

如果没有读取任何字节就到达了流的末尾，则返回 -1。

下面的语句使用 `while` 循环来读取 `FileInputStream` 对象 `diskfile` 中的数据：

```
int newByte = 0;
while (newByte != -1) {
    newByte = diskfile.read();
    System.out.print(newByte + " ");
}
```

该循环每次从 `diskfile` 指向的文件中读取一个字节，将其显示出来并在后面加一个空格，直到达到文件尾。到达文件尾时，将显示 -1——可以使用 `if` 语句来判断是否到达文件尾。

程序清单 15.1 中的应用程序 `ByteReader` 使用类似的技术来读取文件输入流。读取文件的最后一个字节后，调用 `close()` 方法来关闭这个流。不再需要流时，应关闭它，这样将释放系统资源。

#### 程序清单 15.1 完整的 `ByteReader.java` 源代码

```
1: import java.io.*;
2:
3: public class ByteReader {
4:     public static void main(String[] arguments) {
5:         try {
6:             FileInputStream file = new
7:                 FileInputStream("class.dat");
8:             boolean eof = false;
```

```

9:         int count = 0;
10:        while (!eof) {
11:            int input = file.read();
12:            System.out.print(input + " ");
13:            if (input == -1)
14:                eof = true;
15:            else
16:                count++;
17:        }
18:        file.close();
19:        System.out.println("\nBytes read: " + count);
20:    } catch (IOException e) {
21:        System.out.println("Error - " + e.toString());
22:    }
23: }
24: }

```

如果您运行该程序，将出现下述错误消息：

```
Error - java.io.FileNotFoundException: class.dat (The system
cannot find the file specified).
```

上述错误消息类似于编译器产生的异常，但它实际上是由应用程序 `ByteReader` 的第 20~22 行的 `catch` 块生成的。由于找不到文件 `class.dat`，因此第 6 行和第 7 行将引发该异常。

必须有一个用来读取的字节文件。这可以是任何文件，虽然使用较大的文件时，程序需要运行一段时间才能结束。一种不错的选择是程序的类文件，它包含 Java 虚拟机执行的字节码指令。请复制 `ByteReader.class`，并将其重命名为 `class.dat`，以创建一个这样的字节文件。请不要对 `ByteReader.class` 进行重命名，否则将无法运行该程序。

#### 提示

Windows 用户可以在命令行窗口中使用 MS-DOS 来创建 `class.dat`。切换到 `ByteReader.class` 所在的文件夹，然后使用下述命令：

```
copy ByteReader.class class.dat
```

Linux 用户可在命令行执行如下命令：

```
cp ByteReader.class class.dat
```

运行该程序时，它将显示 `class.dat` 中的所有字节，然后显示字节数。如果您使用 `ByteReader.class` 来创建 `class.dat`，则输出的最后几行应是这样的：

```

12 0 50 0 13 0 56 0 14 0 61 0 16 0 64 0 17 0 67 0 18 0 71 0 19 0
96 0 22 0 99 0 20 0 100 0 21 0 128 0 23 0 28 0 0 0 32 0 6 254 0
14 7 0 29 1 1 252 0 46 1 250 0 2 2 255 0 31 0 1 7 0 30 0 1 7 0 31
28 0 1 0 32 0 0 0 2 0 33 -1
Bytes read: 1047

```

在输出中，每行的字节数取决于系统的文本列宽。显示的字节取决于用来创建 `class.dat` 的文件。

## 2. 文件输出流

文件输出流可使用构造函数 `FileOutputStream(String)` 来创建，其用法与构造函数 `FileInputStream(String)` 相同，因此您可以在文件名中指定路径。

指定与输出流相关联的文件时，必须非常小心。如果它与现有的某个文件同名，则当您把数据写入到流中时，原来的数据将被覆盖。

可以使用构造函数 `FileOutputStream(String, boolean)` 来创建这样的文件输出流，即将数据追加到文件末尾。其中的字符串指定了文件，为追加数据而不是覆盖已有的数据，必须将 `boolean` 参数设置为 `true`。

文件输出流的 `write(int)` 方法用于将字节写入到流中。将最后一个字节写入到文件中后，应调用流的 `close()` 方法来关闭它。

要写入多个字节，可使用方法 `write(byte[], int, int)`，其工作原理与前面介绍的 `read(byte[], int, int)` 方法类似。其中各个参数分别是包含要输出的字节的字节数组、该数组的起点以及要写入的字节数。

程序清单 15.2 中的应用程序 ByteWriter 将一个整数数组写入到文件输出流中。

程序清单 15.2 完整的 ByteWriter.java 源代码

```
1: import java.io.*;
2:
3: public class ByteWriter {
4:     public static void main(String[] arguments) {
5:         int[] data = { 71, 73, 70, 56, 57, 97, 13, 0, 12, 0, 145, 0,
6:             0, 255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 44, 0,
7:             0, 0, 0, 13, 0, 12, 0, 0, 2, 38, 132, 45, 121, 11, 25,
8:             175, 150, 120, 20, 162, 132, 51, 110, 106, 239, 22, 8,
9:             160, 56, 137, 96, 72, 77, 33, 130, 86, 37, 219, 182, 230,
10:            137, 89, 82, 181, 50, 220, 103, 20, 0, 59 };
11:         try {
12:             FileOutputStream file = new
13:                 FileOutputStream("pic.gif");
14:             for (int i = 0; i < data.length; i++)
15:                 file.write(data[i]);
16:             file.close();
17:         } catch (IOException e) {
18:             System.out.println("Error - " + e.toString());
19:         }
20:     }
21: }
```

该程序执行的操作如下。

- 第 5~10 行：创建一个名为 data 的整数数组，该数组包含 66 个元素。
- 第 12 和 13 行：使用一个名为 pic.gif 的文件创建一个输出流，该文件位于 ByteWriter.class 所在的文件夹中。
- 第 14~15 行：使用一个 for 循环来遍历 data 数组，并将其中的每个元素写入到文件流中。
- 第 16 行：关闭文件输出流。

运行该程序时，可以使用任何 Web 浏览器或图形编辑工具来打开文件 pic.gif。它是一个小型的 GIF 格式的图像文件，如图 15.1 所示。

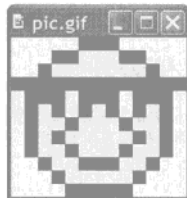


图 15.1 文件 pic.gif (放大后)

## 15.3 过滤流

过滤流对通过现有流传递的信息进行修改，它是使用 FilterInputStream 或 FilterOutputStream 的子类来创建的。

这些类本身并不处理任何过滤操作，它们有诸如 BufferInputStream 和 DataOutputStream 等子类，能够处理特定的过滤类型。

### 字节过滤器

如果以大块的方式传递信息，速度将更快，即使接收这些块的速度高于处理它们的速度。例如，请考虑下述哪种读书方式的速度更快：

- 朋友将整本书借给您阅读;
- 朋友每次借给您一页,您读完后,再借给您另一页。

显然,第一种方式的速度更快,效率也更高。在 Java 中,缓冲流也有这样的优点。

缓冲区是一片存储空间,在读写数据的程序需要之前,数据被存储在这里。使用缓冲区,无需每次都到数据源那里去获取数据。

### 1. 缓冲流

缓冲输入流使用未被处理的数据来填充缓冲区,程序需要数据时,将首先在缓冲区中查找,如果没有找到,再到流源中查找。

缓冲字节流是使用 `BufferedInputStream` 和 `BufferedOutputStream` 类表示的。

要创建缓冲输入流,可使用下述两个构造函数之一。

- `BufferedInputStream(InputStream)`: 为指定的 `InputStream` 对象创建一个缓冲输入流。
- `BufferedInputStream(InputStream, int)`: 为指定的 `InputStream` 创建一个缓冲区大小为 `int` 的缓冲输入流。

要从缓冲输入流中读取数据,最简单的方式是调用其 `read()` 方法,且不提供任何参数。该方法通常返回一个 0~255 的整数,它表示流中的下一个字节,如果到达了流尾,则返回 -1。

像其他输入流一样,您也可以调用 `read(byte[], int, int)` 方法,它将流数据存储到一个字节数组中。

要创建缓冲输出流,可使用下述两个构造函数之一。

- `BufferedOutputStream(OutputStream)`: 为指定的 `OutputStream` 对象创建一个缓冲输出流。
- `BufferedOutputStream(OutputStream, int)`: 为指定的 `OutputStream` 对象创建一个缓冲区大小为 `int` 的缓冲输出流。

可使用输出流的 `write(int)` 方法将单个字节发送到流中。方法 `write(byte[], int, int)` 将指定的字节数组中的多个字节写入到输出流中,其中的参数分别是字节数组、数组的起点和要写入的字节数。

#### 注意

虽然方法 `write()` 接受一个整数作为参数,但其值必须在 0~255 之间。如果指定的参数值大于 255,则实际存储的值将是指定的值除以 256 后的余数。运行本节后面的程序时,您可以测试这一点。

数据被传递到缓冲流中后,在缓冲流已满或缓冲流的 `flush()` 方法被调用之前,数据将不会被输出到目的地。

下面的工程(应用程序 `BufferDemo`) 将一系列的字节写入到一个与文本文件相关联的缓冲输出流中。其中的第一个和最后一个整数是通过命令行参数指定的,如下面的 JDK 命令所示:

```
java BufferDemo 7 64
```

将数据写入文本文件后, `BufferDemo` 使用该文件创建了一个缓冲输入流,并读取其中的字节。该程序的源代码如程序清单 15.3 所示。

程序清单 15.3 完整的 `BufferDemo.java` 源代码

```
1: import java.io.*;
2:
3: public class BufferDemo {
4:     public static void main(String[] arguments) {
5:         int start = 0;
6:         int finish = 255;
7:         if (arguments.length > 1) {
8:             start = Integer.parseInt(arguments[0]);
9:             finish = Integer.parseInt(arguments[1]);
10:        } else if (arguments.length > 0)
11:            start = Integer.parseInt(arguments[0]);
12:        ArgStream as = new ArgStream(start, finish);
```



```

13:         System.out.println("\nWriting: ");
14:         boolean success = as.writeStream();
15:         System.out.println("\nReading: ");
16:         boolean readSuccess = as.readStream();
17:     }
18: }
19:
20: class ArgStream {
21:     int start = 0;
22:     int finish = 255;
23:
24:     ArgStream(int st, int fin) {
25:         start = st;
26:         finish = fin;
27:     }
28:
29:     boolean writeStream() {
30:         try {
31:             FileOutputStream file = new
32:                 FileOutputStream("numbers.dat");
33:             BufferedOutputStream buff = new
34:                 BufferedOutputStream(file);
35:             for (int out = start; out <= finish; out++) {
36:                 buff.write(out);
37:                 System.out.print(" " + out);
38:             }
39:             buff.close();
40:             return true;
41:         } catch (IOException e) {
42:             System.out.println("Exception: " + e.getMessage());
43:             return false;
44:         }
45:     }
46:
47:     boolean readStream() {
48:         try {
49:             FileInputStream file = new
50:                 FileInputStream("numbers.dat");
51:             BufferedInputStream buff = new
52:                 BufferedInputStream(file);
53:             int in = 0;
54:             do {
55:                 in = buff.read();
56:                 if (in != -1)
57:                     System.out.print(" " + in);
58:             } while (in != -1);
59:             buff.close();
60:             return true;
61:         } catch (IOException e) {
62:             System.out.println("Exception: " + e.getMessage());
63:             return false;
64:         }
65:     }
66: }

```

该程序的输出取决于运行它时在命令行中指定的两个参数。如果参数为 4 和 13，则输出如下：

```

Writing:
 4 5 6 7 8 9 10 11 12 13
Reading:
 4 5 6 7 8 9 10 11 12 13

```

该应用程序由两个类构成：BufferDemo 和助手类 ArgStream。BufferDemo 读取两个参数的值（如果用户提供了），并在构造函数 ArgStream() 中使用它们。

第 14 行调用 ArgStream 的 writeStream() 方法，将一系列的字节写入到一个缓冲输出流中，然后第 16 行调用方法 readStream()，读取这些字节。

虽然 writeStream() 和 readStream() 沿两个不同的方向移动数据，但它们几乎是相同的。它们的格

式如下:

- 使用文件名 numbers.dat 来创建一个输入或输出流;
- 使用文件流来创建一个缓冲输入或输出流;
- 使用缓冲流的 write() 方法来发送数据或使用其 read() 方法来读取数据;
- 关闭缓冲流。

由于发生错误时,文件流和缓冲流将引发 IOException 异常,所以所有涉及流的操作都放在 try-catch 中,以捕获并处理这种异常。

#### 提示

writeStream() 和 readStream() 返回的布尔值指出流操作是否成功。在这个程序中,没有使用它们,但让这些方法的调用者知道是否发生了错误是种不错的做法。

## 2. 控制台输入流

很多经验丰富的程序员学习 Java 时怀念的功能之一是,在运行应用程序时,从控制台读取文本或数字输入。并没有与输出方法 System.out.print() 和 System.out.println() 对应的输入方法。

有了缓冲输入流后,您可以使用它来读取控制台输入。

System 类(位于 java.lang 包中)有一个名为 in 的类变量,这是一个 InputStream 对象。该对象通过流从键盘读取输入。

可以像使用其他输入流那样使用这个流。下面的语句创建一个与输入流 System.in 相关联的缓冲输入流:

```
BufferedInputStream command = new BufferedInputStream(System.in);
```

下面的工程(ConsoleInput 类)包含一个类方法,您可以在任何 Java 应用程序中使用它来读取控制台输入。请在编辑器中输入程序清单 15.4 中的代码,然后将其保存为文件 ConsoleInput.java。

程序清单 15.4 完整的 ConsoleInput.java 源代码

```
1: import java.io.*;
2:
3: public class ConsoleInput {
4:     public static String readLine() {
5:         StringBuffer response = new StringBuffer();
6:         try {
7:             BufferedInputStream buff = new
8:                 BufferedInputStream(System.in);
9:             int in = 0;
10:            char inChar;
11:            do {
12:                in = buff.read();
13:                inChar = (char) in;
14:                if ((in != -1) & (in != '\n') & (in != '\r')) {
15:                    response.append(inChar);
16:                }
17:            } while ((in != -1) & (inChar != '\n') & (in != '\r'));
18:            buff.close();
19:            return response.toString();
20:        } catch (IOException e) {
21:            System.out.println("Exception: " + e.getMessage());
22:            return null;
23:        }
24:    }
25:
26:    public static void main(String[] arguments) {
27:        System.out.print("\nWhat is your name? ");
28:        String input = ConsoleInput.readLine();
29:        System.out.println("\nHello, " + input);
30:    }
31: }
```

`ConsoleInput` 类有一个 `main()` 方法，它演示了类方法 `readLine()` 的用法。编译这个类，并将其作为应用程序运行时，输出将如下：

```
What is your name? Amerigo Vespucci
```

```
Hello, Amerigo Vespucci
```

`ConsoleInput()` 使用缓冲输入流的 `read()` 方法读取该流中的用户输入，而 `read()` 方法在达到输入末尾时返回-1。输入末尾指的是用户按回车键或者遇到字符‘\r’（回车）或‘\n’（换行）。

### 3. 数据流

要处理未表示为字节或字符的数据，可以使用数据输入流和数据输出流。这些流对字节流进行过滤，以便能够直接读写如下基本数据类型：`boolean`、`byte`、`double`、`float`、`int`、`long` 和 `short`。

要创建数据输入流，可使用构造函数 `DataInputStream(InputStream)`。其中的参数是一个现有的输入流，如一个缓冲输入流或一个文件输入流。

要创建数据输出流，可使用构造函数 `DataOutputStream(OutputStream)`，它指定了相关联的输出流。可用于数据输入和输出流的读写方法如下：

- `readBoolean()`、`writeBoolean(boolean)`;
- `readByte()`、`writeByte(integer)`;
- `readDouble()`、`writeDouble(double)`;
- `readFloat()`、`writeFloat(float)`;
- `readInt()`、`writeInt(int)`;
- `readLong()`、`writeLong(long)`;
- `readShort()`、`writeShort(int)`。

其中每个输入方法都返回相应的基本数据类型。例如，方法 `readFloat()` 返回一个 `float` 值。

还有方法 `readUnsignedByte()` 和 `readUnsignedShort()`，它们用于读取无符号的 `byte` 和 `short` 值。Java 并不支持这些数据类型，因此它们被作为 `int` 值返回。

#### 注意

无符号 `byte` 的取值范围为 0~255，这不同于 Java 的变量类型 `byte`，后者的取值范围为 -128~127。同样，无符号 `short` 的取值范围为 0~65 535，而不是 `short` 类型的 -32 768~32 767。

数据输入流的各种读取方法并不都返回一个指出是否到达流尾的值。

读取方法达到流尾时，将引发 `EOFException`（文件尾异常）。可将读取数据的循环放在 `try` 块中，而对应的 `catch` 块只处理 `EOFException` 异常。您可以在 `catch` 块中调用流的 `close()` 方法，并执行其他的清理工作。

下面的程序演示了这一点。程序清单 15.5 和 15.6 是两个使用数据流的程序。应用程序 `PrimeWriter` 将前 400 个素数作为整数写入到文件 `400primes.dat` 中；应用程序 `PrimeReader` 读取这个文件中的整数，并将它们显示出来。

程序清单 15.5 完整的 `PrimeWriter.java` 源代码

```
1: import java.io.*;
2:
3: public class PrimeWriter {
4:     public static void main(String[] arguments) {
5:         int[] primes = new int[400];
6:         int numPrimes = 0;
7:         // candidate: the number that might be prime
8:         int candidate = 2;
9:         while (numPrimes < 400) {
10:             if (isPrime(candidate)) {
```

```

11:         primes[numPrimes] = candidate;
12:         numPrimes++;
13:     }
14:     candidate++;
15: }
16:
17: try {
18:     // Write output to disk
19:     FileOutputStream file = new
20:         FileOutputStream("400primes.dat");
21:     BufferedOutputStream buff = new
22:         BufferedOutputStream(file);
23:     DataOutputStream data = new
24:         DataOutputStream(buff);
25:
26:     for (int i = 0; i < 400; i++)
27:         data.writeInt(primes[i]);
28:     data.close();
29: } catch (IOException e) {
30:     System.out.println("Error - " + e.toString());
31: }
32: }
33:
34: public static boolean isPrime(int checkNumber) {
35:     double root = Math.sqrt(checkNumber);
36:     for (int i = 2; i <= root; i++) {
37:         if (checkNumber % i == 0)
38:             return false;
39:     }
40:     return true;
41: }
42: }

```

#### 程序清单 15.6 完整的 PrimeReader.java 源代码

```

1: import java.io.*;
2:
3: public class PrimeReader {
4:     public static void main(String[] arguments) {
5:         try {
6:             FileInputStream file = new
7:                 FileInputStream("400primes.dat");
8:             BufferedInputStream buff = new
9:                 BufferedInputStream(file);
10:            DataInputStream data = new
11:                DataInputStream(buff);
12:
13:            try {
14:                while (true) {
15:                    int in = data.readInt();
16:                    System.out.print(in + " ");
17:                }
18:            } catch (EOFException eof) {
19:                buff.close();
20:            }
21:        } catch (IOException e) {
22:            System.out.println("Error - " + e.toString());
23:        }
24:    }
25: }

```

应用程序 PrimeWriter 的大部分代码用于查找前 400 个素数。有了包含前 400 个素数的整数数组后，第 17~31 行将其写入到一个数据输出流中。

该应用程序对一个流使用了多个过滤器。开发这个流的步骤为：

1. 创建一个与文件 400primes.dat 相关联的文件输出流；
2. 将一个缓冲输出流与文件流关联起来；
3. 将一个数据输出流与缓冲流关联起来。

数据流的 `writeInt()` 方法被用来将素数写入到文件中。

应用程序 `PrimeReader` 更简单,因为它不需要执行任何与素数有关的操作,而只是使用数据输入流读取文件中的整数。

`PrimeReader` 的第 6~11 行与应用程序 `PrimeWriter` 中的语句几乎相同,只是使用的是输入类,而不是输出类。

处理 `EOFException` 异常的 `try-catch` 块位于第 13~20 行。读取数据的操作是在 `try` 块中完成的。

语句 `while(true)` 创建了一个死循环。这并没有什么问题:随着不断地读取数据流,将到达流尾,这将引发 `EOFException` 异常。第 15 行中的 `readInt()` 方法从流中读取整数。

应用程序 `PrimeReader` 的最后几行输出如下:

```
2137 2141 2143 2153 2161 2179 2203 2207 2213 2221 2237 2239 2243 22
51 2267 2269 2273 2281 2287 2293 2297 2309 2311 2333 2339 2341 2347
2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423 2437 2
441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 255
7 2579 2591 2593 2609 2617 2621 2633 2647 2657 2659 2663 2671 2677
2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741
```

## 15.4 字符流

知道如何处理字节流后,便掌握了处理字符流所需的大部分技能。字符流用于处理用 ASCII 字符集或 Unicode (包含 ASCII 的国际字符集) 表示的文本。

可以使用字符流来处理的文件有纯文本文件、超文本标记语言 (HTML) 文档和 Java 源代码文件。

用于读写这些流的类都是 `Reader` 和 `Writer` 的子类。对于所有的文本输入,都应使用字符流来处理,而不能直接使用字节流来处理。

### 15.4.1 读取文本文件

从文件中读取字符流时,通常使用 `FileReader` 类。这个类是从 `InputStreamReader` 派生而来的,它读取字节流中的字节,并将其转换为表示 Unicode 字符的整数值。

要将字符输入流与文件关联起来,可使用构造函数 `FileReader(String)`。字符串参数指定了要关联的文件,除文件名外,它还可以包含路径。

下面的语句创建一个名为 `look` 的 `FileReader` 对象,并将其与文本文件 `index.txt` 关联起来:

```
FileReader look = new FileReader("index.txt");
```

有了 `FileReader` 后,可以用它的下述方法来读取文件中的字符。

- `read()`: 将流中的下一个字符作为整数返回。
- `read(char[], int, int)`: 将指定数目的字符读入到指定字符数组的指定位置。

第二个方法的工作原理与字节流输入类中的相应方法类似。它并不返回下一个字符,而是返回读取的字符数或 -1 (如果没有读取任何字符就到达了流尾)。

下面的方法使用 `FileReader` 对象 `text` 来读取一个文本文件中的字符,并将其显示出来:

```
FileReader text = new FileReader("readme.txt");
int inByte;
do {
    inByte = text.read();
    if (inByte != -1)
        System.out.print( (char)inByte );
} while (inByte != -1);
System.out.println("");
text.close();
```

由于字符流的 `read()` 方法返回一个整数,所以必须首先将它强制转换为字符,然后才能显示它、将它存储到数组中或使用它来构成一个字符串。每个字符都有一个对应的数值编码,它表示该字符在

Unicode 字符集中的位置。从流中读取的整数就是数值编码。

要一次读取整行文本，而不是一个字符，可以结合使用 `FileReader` 和 `BufferedReader` 类。

`BufferedReader` 类读取字符输入流，并将读取的字符存储到缓冲区，以提高效率。要创建缓冲版本，必须有 `Reader` 对象。下面的构造函数可用于创建 `BufferedReader`。

- `BufferedReader(Reader)`：创建与指定的 `Reader` 对象（如 `FileReader`）相关联的缓冲字符流。
- `BufferedReader(Reader, int)`：创建与指定的 `Reader` 对象相关联的缓冲字符流，其缓冲区大小为 `int`。

缓冲字符流可使用方法 `read()` 和 `read(char[], int, int)` 来读取，要读取一行文本，可以使用方法 `readLine()`。

方法 `readLine()` 返回一个 `String` 对象，其中包含流中的下一行文本，但不包括表示行尾的字符。如果到达流尾，则返回 `null`。

行尾是通过下列字符来标识的：

- 换行符（`'\n'`）；
- 回车符（`'\r'`）；
- 回车符和换行符（`"\n\r"`）。

程序清单 15.7 中的 Java 应用程序通过缓冲字符流来读取其源代码。

程序清单 15.7 完整的 `SourceReader.java` 源代码

```
1: import java.io.*;
2:
3: public class SourceReader {
4:     public static void main(String[] arguments) {
5:         try {
6:             FileReader file = new
7:                 FileReader("SourceReader.java");
8:             BufferedReader buff = new
9:                 BufferedReader(file);
10:            boolean eof = false;
11:            while (!eof) {
12:                String line = buff.readLine();
13:                if (line == null)
14:                    eof = true;
15:                else
16:                    System.out.println(line);
17:            }
18:            buff.close();
19:        } catch (IOException e) {
20:            System.out.println("Error - " + e.toString());
21:        }
22:    }
23: }
```

该程序中的大部分内容与本章前面的程序类似。

- 第 6 和 7 行：创建一个输入源——与文件 `SourceReader.java` 相关联的 `FileReader` 对象。
- 第 8 和 9 行：将一个缓冲过滤器（`BufferedReader` 对象 `buff`）与输入源关联起来。
- 第 11~17 行：在 `while` 循环中使用 `readLine()` 方法每次读取文本文件中的一行，当该方法返回 `null` 时，循环结束。

应用程序 `SourceReader` 的输出为文本文件 `SourceReader.java` 的内容。

## 15.4.2 写文本文件

类 `FileWriter` 用于将字符流写入到文件中，它是 `OutputStreamWriter` 的子类，后者有一些将 Unicode 编码转换为字节的行为。

`FileWriter` 构造函数有两个：`FileWriter(String)`和 `FileWriter(String, boolean)`。在此，字符串指定了字符流将被写入到哪个文件中，其中可以包含路径。如果要将数据追加到文本文件末尾，可将可选的布尔参数设置为 `true`。与其他流写入类一样，必须避免在追加数据时覆盖已有的数据。

可用于将数据写入流中的 `FileWriter` 方法如下。

- `write(int)`：写入一个字符。
- `write(char[], int, int)`：从指定位置开始，写入指定字符数组中指定数目的字符。
- `write(String, int, int)`：从指定位置开始，写入指定字符串中指定数目的字符。

下面的例子使用 `FileWriter` 类和 `write(int)`方法将字节流写入到文件中：

```
FileWriter letters = new FileWriter("alphabet.txt");
FileWriter letters = new FileWriter("alphabet.txt");
for (int i = 65; i < 91; i++)
    letters.write( (char)i );
letters.close();
```

将所有的字符都发送到目标文件后，使用方法 `close()`关闭流。上述代码生成的 `alphabet.txt` 文件如下：

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

类 `BufferedWriter` 可用于写缓冲字符流。要创建这个类的对象，可使用构造函数 `BufferedWriter(Writer)`或 `BufferedWriter(Writer, int)`。`Writer` 参数可以是任何字符输出流类，如 `FileWriter`。第二个参数是可选的，它是一个整数，指出了要使用的缓冲区大小。

`BufferedWriter` 有 3 个与 `FileWriter` 相同的输出方法：`write(int)`、`write(char[], int, int)`和 `write(String, int, int)`。

另一个有用的输出方法是 `newLine()`，它发送运行程序的平台使用的行尾字符。

#### 提示

在将文件从一种操作系统传输到另一种操作系统中（如当 Windows XP 用户将文件上传到运行在 Linux 操作系统下的 Web 服务器上）时，不同的行尾标记可能导致转换问题。使用 `newLine()`而不是字面量（如“`\n`”），可使程序在跨平台时对用户更友好。

方法 `close()`用于关闭缓冲字符流，并确保将所有被缓冲数据都发送到流的目的地。

## 15.5 文件和文件名过滤器

到目前为止的所有示例都使用一个字符串来指定流操作涉及的文件。通常，对于使用文件和流的程序而言，这足够了；但如果要复制文件、重命名文件或处理其他任务，则可以使用 `File` 对象。

`File` 类也位于 `java.io` 包中，它表示文件或文件夹引用。可使用的 `File` 构造函数如下。

- `File(String)`：使用指定的文件夹创建一个 `File` 对象。没有指定文件名，因此这是文件夹引用。
- `File(String, String)`：使用指定的文件夹路径和文件名创建一个 `File` 对象。
- `File(File, String)`：创建一个 `File` 对象，其路径由 `File` 指定，文件名由 `String` 指定。

可以对 `File` 对象调用一些有用的方法。

方法 `exists()`返回一个布尔值，指出创建 `File` 对象时，指定文件夹中的文件是否存在。如果存在，则可以调用方法 `length()`，该方法返回一个 `long` 值，指出该文件的长度（单位为字节）。

方法 `renameTo(File)`将文件名重命名为 `File` 参数所指定的名称。它返回一个布尔值，指出操作是否成功。

要删除文件或文件夹，可调用方法 `delete()`或 `deleteOnExit()`。方法 `delete()`立刻进行删除操作（返回一个布尔值，指出是否成功删除）；方法 `deleteOnExit()`在程序结束时删除操作，它不返回任何值，为使该方法管用，程序必须结束。

方法 `getName()` 和 `getPath` 返回包含文件名和路径的字符串。

当 `File` 对象表示一个文件夹而不是文件时，有几个方法很有用。

方法 `mkdir()` 用于创建一个文件夹，该文件夹是由方法所属的 `File` 对象指定的。它返回一个布尔值，指出是否成功。没有与之对应的删除文件夹的方法，因为 `delete()` 可用于删除文件和文件夹。

如果 `File` 对象是文件夹，则方法 `isDirectory()` 返回 `true`，否则返回 `false`。

方法 `listFiles()` 返回一个 `File` 对象数组，其中包含文件夹中的所有文件和子文件夹。

与其他文件处理操作一样，使用这些方法时必须小心，以避免误删文件和文件夹或损坏数据。没有用于恢复被删除的文件或文件夹的方法。

如果程序没有相应的安全机制来执行文件操作，上述方法都将引发 `SecurityException`，因此必须使用 `try-catch` 块或在方法声明中使用 `throws` 子句来处理这些异常。

程序清单 15.8 中的程序将文件中所有的文本转换为大写字符。它使用缓冲输入流来读取文件，每次读取一个字符。将字符转换为大写后，使用缓冲输出流将其发送到临时文件中。这里使用 `File` 对象（而不是字符串）来指出涉及的文件，因此可以在需要时删除文件 and 对其进行重命名。

程序清单 15.8 完整的 `AllcapsDemo.java` 源代码

```

1: import java.io.*;
2:
3: public class AllCapsDemo {
4:     public static void main(String[] arguments) {
5:         AllCaps cap = new AllCaps(arguments[0]);
6:         cap.convert();
7:     }
8: }
9:
10: class AllCaps {
11:     String sourceName;
12:
13:     AllCaps(String sourceArg) {
14:         sourceName = sourceArg;
15:     }
16:
17:     void convert() {
18:         try {
19:             // Create file objects
20:             File source = new File(sourceName);
21:             File temp = new File("cap" + sourceName + ".tmp");
22:
23:             // Create input stream
24:             FileReader fr = new
25:                 FileReader(source);
26:             BufferedReader in = new
27:                 BufferedReader(fr);
28:
29:             // Create output stream
30:             FileWriter fw = new
31:                 FileWriter(temp);
32:             BufferedWriter out = new
33:                 BufferedWriter(fw);
34:
35:             boolean eof = false;
36:             int inChar = 0;
37:             do {
38:                 inChar = in.read();
39:                 if (inChar != -1) {
40:                     char outChar = Character.toUpperCase( (char)inChar );
41:                     out.write(outChar);
42:                 } else
43:                     eof = true;
44:             } while (!eof);
45:             in.close();
46:             out.close();
47:         }

```



```

48:         boolean deleted = source.delete();
49:         if (deleted)
50:             temp.renameTo(source);
51:     } catch (IOException e) {
52:         System.out.println("Error - " + e.toString());
53:     } catch (SecurityException se) {
54:         System.out.println("Error - " + se.toString());
55:     }
56: }
57: }

```

编译该程序后，需要一个可被转换为大写的文本文件。办法之一是复制 AllCapsDemo.java，并将其命名为 TempFile.java。

要转换的文件在运行 AllCapsDemo 时通过命令行指定，如下述 JDK 示例所示：

```
java AllCapsDemo TempFile.java
```

该程序不产生任何输出。请在文本编辑器打开转换后的文件，以查看该应用程序的结果。

## 15.6 总结

本章介绍了如何沿两个不同的方向来处理流：通过输入流将数据送入程序中和使用输出流将数据发送出去。

使用字节流来处理很多类型的非文本数据，使用字符流来处理文本。要改变通过流发送信息的方式或修改信息本身，可以将过滤器与流关联起来。

除本章介绍的类外，java.io 包还提供了您可能想了解的其他类型的流。在不同的线程间传送数据时，管道流很有用，而字节数组流可将程序与计算机相连。

由于 Java 中的流类都很相似，因此您具备了使用其他类型所需的大部分知识。它们的构造函数、读取方法和写入方法几乎都相同。

流是扩展 Java 程序功能的重要途径，因为它们提供了到各种数据类型的连接。

下一章将使用流来读写 Java 对象。

## 15.7 问与答

问：我使用 C 程序创建了一个由整数和其他数据组成的文件，可以使用 Java 程序来读取该文件吗？

答：可以，必须考虑的因素之一是，C 程序表示整数的方式是否与 Java 程序相同。您可能还记得，所有的数据都被表示为单个字节或一系列的字节。在 Java 中，使用 4 个字节来表示整数，这些字节以 big-endian 顺序排列。可以从左到右地将这些字节组合起来确定整数的值。Intel PC 上实现的 C 程序很可能以 little-endian 顺序来表示整数，这意味着必须从右到左地组合字节来确定整数值。要使用 Java 之外的编程语言创建数据文件，您可能需要学习一些高级技术，如移位。

问：在 Java 中指定文件名时可以使用相对路径吗？

答：相对路径是根据当前的用户文件夹确定的，后者存储在系统属性 user.dir 中。可以使用 java.lang 包中的 System 类来获悉到该文件夹的完整路径，这个包不需要导入。

为此，可以调用 System 的类方法 getProperty(String)，并将这个系统属性的名称作为参数，如下例所示：

```
String userFolder = System.getProperty("user.dir");
```

这个方法以字符串的方式返回路径。

问：FileWriter 类有一个 write(int) 方法，用于将字符发送到文件中。该方法为何不是 write(char) 呢？

答：在很多方面，数据类型 char 和 int 是可互换的：可以在需要 char 的方法中使用 int，反之亦然。这是由于每个字符都被表示为一个数值编码（整数值）。当您使用 int 值作为参数调用 write() 方法时，它将返回该 int 值对应的字符。调用 write() 方法时，可以将 int 值强制转换为 char 值，以确保该方法按您的意愿被使用。

## 15.8 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 15.8.1 问题

1. 如果创建 FileOutputStream 时使用已有的文件的引用，将发生什么情况？
  - a. 引发异常；
  - b. 写入到流中的数据将被追加到该文件的末尾；
  - c. 该文件的内容将被替换为写入到流中的数据。
2. 处理流时，哪两种基本数据类型可以互换？
  - a. byte 和 boolean；
  - b. char 和 int；
  - c. byte 和 char。
3. 在 Java 中，byte 变量和无符号 byte 变量的最大值分别是多少？
  - a. 都是 255；
  - b. 都是 127；
  - c. 前者为 127，后者为 255。

答案

1. c，这是使用输出流时应注意的问题之一：一不小心就可能覆盖已有文件的内容。
2. b，由于在 Java 内部，char 被表示为整数值，因此在方法调用和其他语句中，这两种类型可以互换。
3. c，基本数据类型 byte 的取值范围为 -128 ~ 127，而无符号 byte 的取值范围为 0 ~ 255。

### 15.8.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

对于下述代码：

```
import java.io.*;

public class Unknown {
    public static void main(String[] arguments) {
        String command = "";
        BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
        try {
            command = br.readLine();
        }
    }
}
```

```
        catch (IOException e) { }  
    }  
}
```

该程序能够成功地将一行控制台输入存储到名为 `command` 的 `String` 对象中吗？

- a. 能够；
- b. 不能，因为要读取控制台输入，必须使用缓冲输入流；
- c. 不能，因为该程序无法通过编译；
- d. 不能，因为它读取了多行控制台输入。

答案 a

## 15.9 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 请修改第 7 章中的 `HexRead` 程序，使之从文本文件中读取两位的十六进制序列，并显示相应的十进制值。
2. 编写一个程序，它读取一个文件以判断其中的字节数，然后使用 0 覆盖所有的字节（出于显而易见的原因，请不要使用要保留的文件来测试该程序，否则该文件中的所有数据都将被破坏）。



## 第 16 章

# 序列化和查看对象

面向对象编程的一个关键概念是数据的表示方式。在面向对象语言（如 Java）中，对象表示两方面的内容。

- 行为：对象的功能。
- 属性：将对象与其他对象区分开来的数据。

将行为和属性组合起来是面向对象语言不同于其他编程语言的地方之一。使用其他编程语言时，程序通常是一组操作数据的指令，而数据独立于程序，就像字处理软件中那样。大多数字处理器都被视为用于创建和编辑文本文件的程序。

面向对象编程和其他技术使程序和数据间的界线不再清晰。在诸如 Java 等语言中，对象封装了指令（行为）和数据（属性）。

本章将介绍 Java 程序利用这种表示方式的两种途径。

- 对象序列化：使用流来读写对象。
- 反射：对象了解另一个对象的细节。

### 16.1 对象序列化

正如第 15 章介绍的，Java 通过流来访问外部数据。流是一种将数据从一个地方传送到另一个地方的对象。有些流将数据从源传递到 Java 程序中；其他流则沿相反的方向传送数据，即将数据从程序传递到目的地。

将网页上的数据读入到 Java 程序的数组中的流属于前者；而将 String 数组写入到磁盘文件中的流则属于后者。

在第 15 章中，介绍了两种流。

- 字节流：读写 0~255 的整数值。
- 字符流：读写文本数据。

这些流将数据同处理它们的 Java 类分开。要在以后使用这些数据，必须通过流来读取它并将其转换为类可以使用的格式，如一系列的基本数据类型或对象。

第三种流——对象流，使得可以将数据表示为对象，而不是外部格式。

对象流与字节流和字符流一样，也位于 java.io 包中。要使用它们，必须采用第 15 章介绍的很多技术。

要将对象保存到目的地（如磁盘文件），必须将其转换为串行格式。

#### 注意

串行数据每次被发送一个元素，就像装配线上的汽车。您可能熟悉计算机的串口，它将数据信息作为一系列的位依次发送出去。另一种发送数据的方式是用并行，即同时传输多个元素。

对象通过实现 `Serializable` 接口来指出它可用于流。`Serializable` 接口位于 `java.io` 包中，其使用方式不同于其他接口：它不包含任何实现它的类必须实现的方法。`Serializable` 的唯一用途是，指出类的对象可以以串行方式被存储和检索。

对象可被序列化到硬盘中，也可跨越网络（如 Internet）被序列化，甚至可以涉及不同的操作系统。您可以在 Windows 机器上创建一个对象，将它序列化到 Linux 机器中，然后将它装载到最初的 Windows 机器中，而不会导致任何错误。对象被序列化时，Java 透明地处理用于在这些系统存储数据的不同格式。

对象序列化涉及的一个编程概念是永久性（persistence）：对象在创建它的程序外仍能够存在和起作用。

通常情况下，未被序列化的对象不是永久性的。使用对象的程序终止后，该对象也将不复存在。

序列化让对象能够永久，因为即使没有 Java 程序运行，被存储的对象仍然提供某种用途。被存储的对象包含了可在程序中恢复的信息，使之能够继续发挥作用。

当对象以串行格式被保存到流中时，其引用的所有对象也将被保存。这使得处理序列化更容易，您可以使用一个对象流来维护多个对象。

如果有多个对象包含指向同一个对象的引用，Java 将确保只有对象的一个拷贝被序列化。每个对象都被分配一个内部序列号，随后保存该对象时将只存储相应的序列号。

您还可以将对象的某些变量排除在序列化之外，这可能是为了节省磁盘空间或防止可能带来安全风险的信息被保存。本章后面将介绍到，这需要使用限定符 `transient`。

### 16.1.1 对象输出流

对象是通过类 `ObjectOutputStream` 被写入到流中的。

要创建对象输出流，可使用构造函数 `ObjectOutputStream(OutputStream)`。其中的参数可以是：

- 一个输出流：指出了对象将以串行格式被存储到的目的地。
- 一个过滤器：与指向目的地的输出流相关联。

与其他流一样，也可以在输出流和对象输出流之间串接多个过滤器。

下面的代码创建一个输出流和一个与之相关联的对象输出流：

```
FileOutputStream disk = new FileOutputStream(
    "SavedObject.dat");
ObjectOutputStream disk0 = new ObjectOutputStream(disk);
```

这里创建的对象输出流名为 `disk0`。可以使用 `ObjectOutputStream` 的方法将可序列化的对象和其他信息写入到文件 `SaveObject.dat` 中。

创建对象输出流后，可以调用其 `writeObject(Object)` 方法将对象写入到这个流中。

下面的语句调用前面创建的 `disk0` 流的这个方法：

```
disk0.writeObject(userData);
```

上述语句将对象 `userData` 写入到 `disk0` 对象输出流中。要使上述语句正常工作，`userData` 所属的类必须是可序列化的。

也可以使用对象输出流的下述方法来写入其他类型的信息。

- `write(int)`：将指定的整数写入到流中，该整数必须在 0~255 的范围之内。
- `write(byte[])`：写入指定的字节数组。
- `write(byte[], int, int)`：写入指定字节数组的子集。第二个参数指出了要写入的第一个数组元素，最后一个参数指出了要写入的元素个数。
- `writeBoolean(boolean)`：写入指定的 `boolean` 值。
- `writeBytes(int)`：将指定的整数值作为字节值写入。

- `writeBytes(String)`: 将指定的字符串作为一系列字节写入。
- `writeChar(int)`: 写入指定的字符。
- `writeChars(String)`: 将指定的字符串作为一系列的字符写入。
- `writeDouble(double)`: 写入指定的 `double` 值。
- `writeFloat(float)`: 写入指定的 `float` 值。
- `writeInt(int)`: 写入指定的 `int` 值, 与 `write(int)` 不同, 这个方法的参数可以是任何 `int` 值。
- `writeLong(long)`: 写入指定的 `long` 值。
- `writeShort(short)`: 写入指定的 `short` 值。

构造函数 `ObjectOutputStream` 以及所有将数据写入对象输出流的方法都可能引发 `IOException` 异常。因此必须使用 `try-catch` 块或 `throws` 子句来处理。

程序清单 16.1 中的 Java 应用程序由两个类组成: `ObjectWriter` 和 `Message`。类 `Message` 表示电子邮件。这个类包含对象 `from` 和 `to`, 它们分别用于存储发送方和接收方的姓名。还包含对象 `now` 和数组 `text`, 前者用于存储一个 `Date` 值, 表示发送时间, 后者是一个 `String` 对象数组, 用于存储消息本身。另外, 这个类还包含一个名为 `lineCount` 的 `int` 变量, 用于跟踪消息的行数。

设计发送和接收电子邮件的程序时, 应使用某种流来将消息保存到磁盘中。当消息中的信息从一个地方传递到另一个地方时, 必须以某种格式保存它。这可能需要保存到接收方能够读取为止。

为保留消息, 可将每个消息元素分别保存到字节流或字符流中。在类 `Message` 中, 对象 `from` 和 `to` 可作为字符串被写入到流中, 而 `text` 对象可作为字符串数组被写入。对象 `now` 要复杂些, 因为无法将 `Date` 对象写入到字符流中。然而, 可以将其转换为一系列的整数值, 它们分别表示日期的各个部分: 小时、分、秒等。而整数值可以被写入到流中。

使用对象输出流使得无需首先将 `Message` 对象转换为另一种格式就可以保存它。

程序清单 16.1 中的 `ObjectWriter` 类创建了一个 `Message` 对象、设置其变量的值, 并通过一个对象输出流将它保存到文件 `Message.obj` 中。

#### 程序清单 16.1 完整的 `ObjectWriter.java` 源代码

```

1: import java.io.*;
2: import java.util.*;
3:
4: public class ObjectWriter {
5:     public static void main(String[] arguments) {
6:         Message mess = new Message();
7:         String author = "Sam Wainwright, London";
8:         String recipient = "George Bailey, Bedford Falls";
9:         String[] letter = { "Mr. Gower cabled you need cash. Stop.",
10:             "My office instructed to advance you up to twenty-five",
11:             "thousand dollars. Stop. Hee-haw and Merry Christmas." };
12:         Date now = new Date();
13:         mess.writeMessage(author, recipient, now, letter);
14:         try {
15:             FileOutputStream fo = new FileOutputStream(
16:                 "Message.obj");
17:             ObjectOutputStream oo = new ObjectOutputStream(fo);
18:             oo.writeObject(mess);
19:             oo.close();
20:             System.out.println("Object created successfully.");
21:         } catch (IOException e) {
22:             System.out.println("Error - " + e.toString());
23:         }
24:     }
25: }
26:
27: class Message implements Serializable {
28:     int lineCount;
29:     String from, to;
30:     Date when;

```

```

31:    String[] text;
32:
33:    void writeMessage(String inFrom,
34:        String inTo,
35:        Date inWhen,
36:        String[] inText) {
37:
38:        text = new String[inText.length];
39:        for (int i = 0; i < inText.length; i++)
40:            text[i] = inText[i];
41:        lineCount = inText.length;
42:        to = inTo;
43:        from = inFrom;
44:        when = inWhen;
45:    }
46: }

```

编译并运行应用程序 `ObjectWriter` 时，其输出如下：

Object created successfully.

### 16.1.2 对象输入流

要从流中读取对象，可使用 `ObjectInputStream` 类。与其他流一样，处理对象输入流的方式与处理对象输出流类似，主要区别在于数据的方向不同。

要创建对象输入流，可使用构造函数 `ObjectInputStream(InputStream)`。该构造函数可能引发两种异常：`IOException` 和 `StreamCorruptionException`。`IOException` 是所有的流类都可能引发的异常，这种异常在传输数据过程中发生输入/输出错误时引发。`StreamCorruptionException` 是对象流特有的，它表明流中的数据不是被序列化的对象。

对象输入流可使用输入流或过滤流来创建。

下面的代码创建了一个输入流和一个相应的对象输入流：

```

try {
    FileInputStream disk = new FileInputStream(
        "SavedObject.dat");
    ObjectInputStream obj = new ObjectInputStream(disk);
} catch (IOException ie) {
    System.out.println("IO error -- " + ie.toString());
} catch (StreamCorruptionException se) {
    System.out.println("Error - data not an object.");
}

```

该对象输入流读取文件 `SavedObject.dat` 中的对象的数据。如果该文件不存在或由于某种原因无法读取磁盘，将引发 `IOException` 异常；如果该文件不是一个被序列化的对象，将引发 `StreamCorruptionException` 异常，以指出这一点。

可以使用 `readObject()` 方法从对象输入流中读取对象，该方法返回一个 `Object`。然后，可以将该对象强制转换为它所属的类，如下所示：

```
WorkData dd = (WorkData)disk.readObject();
```

上述语句从对象流 `disk` 中读取一个对象，并将它强制转换为 `WorkData` 类的对象。除 `IOException` 外，该方法还可能引发异常 `OptionalDataException` 和 `ClassNotFoundException`。

`OptionalDataException` 指出流中包含的数据不是被序列化的对象数据，因此无法从流中读取对象。

当从流中读取的对象属于找不到的类时，将引发 `ClassNotFoundException` 异常。对象被序列化时，其所属的类本身并不会被保存到流中。而是将类的名称保存到流中，当从流中装载对象时，Java 解释器将装载这个类。

使用下述的方法可从对象输入流中读取其他类型的信息。

- `read()`: 读取流中的下一个字节, 并返回一个 `int` 值。
- `read(byte[], int, int)`: 将字节读入指定的字节数组。第二个参数指定了用于存储字节的第一个数组元素; 最后一个参数指出了要读取多少个字节。
- `readBoolean()`: 从流中读取一个 `boolean` 值。
- `readByte()`: 从流中读取一个 `byte` 值。
- `readChar()`: 从流中读取一个 `char` 值。
- `readDouble()`: 从流中读取一个 `double` 值。
- `readFloat()`: 从流中读取一个 `float` 值。
- `readInt()`: 从流中读取一个 `int` 值。
- `readLine()`: 从流中读取一个 `String` 值。
- `readLong()`: 从流中读取一个 `long` 值。
- `readShort()`: 从流中读取一个 `short` 值。
- `readUnsignedByte()`: 从流中读取一个无符号 `byte` 值, 并将作为 `int` 值返回。
- `readUnsignedShort()`: 从流中读取一个无符号 `short` 值, 并将它作为 `int` 值返回。

如果在读取流的过程中发生输入/输出错误, 上述方法都将引发 `IOException` 异常。

通过读取对象流来创建对象时, 该对象是完全按照存储在流中的变量和对象信息来创建的。无需调用构造方法来创建变量和设置它们的初值。创建的对象与原来被序列化的对象完全相同。

程序清单 16.2 中的 Java 应用程序从一个流中读取对象, 并将其变量显示到标准输出中。应用程序 `ObjectReader` 装载被序列化到文件 `message.obj` 中的对象。

要运行这个应用程序, 必须首先切换到文件 `message.obj` 和类 `Message` 所在的文件夹中。

#### 程序清单 16.2 完整的 `ObjectReader.java` 源代码

```

1: import java.io.*;
2: import java.util.*;
3:
4: public class ObjectReader {
5:     public static void main(String[] arguments) {
6:         try {
7:             FileInputStream fi = new FileInputStream(
8:                 "message.obj");
9:             ObjectInputStream oi = new ObjectInputStream(fi);
10:            Message mess = (Message) oi.readObject();
11:            System.out.println("Message:\n");
12:            System.out.println("From: " + mess.from);
13:            System.out.println("To: " + mess.to);
14:            System.out.println("Date: " + mess.when + "\n");
15:            for (int i = 0; i < mess.lineCount; i++)
16:                System.out.println(mess.text[i]);
17:            oi.close();
18:        } catch (Exception e) {
19:            System.out.println("Error - " + e.toString());
20:        }
21:    }
22: }

```

该程序的输出如下:

Message:

From: Sam Wainwright, London  
 To: George Bailey, Bedford Falls  
 Date: Sat Jan 13 20:53:40 EST 2007

Mr. Gower cabled you need cash. Stop.  
 My office instructed to advance you up to twenty-five  
 thousand dollars. Stop. Hee-haw and Merry Christmas.



### 16.1.3 暂态变量

创建可被序列化的对象时，需要考虑的一个方面是，是否要保存该对象的所有实例变量。

对于有些实例变量，每当恢复对象时都需要重新创建。一个这样的例子是引用文件或输入流的对象。每次从对象流中读取被序列化的对象时，对于其中的引用文件或输入流的对象，都必须重新创建，因此序列化对象时，没有理由保存这样的信息。

不对包含敏感信息的变量进行序列化是个不错的主意。如果对象中存储了访问某个资源所需的密码，则将密码序列化到文件中将它置于更为危险的境地。如果该密码位于通过跨越网络的流恢复的对象中，则可能被他人检测到。

不序列化变量的另一个原因是，减少存储对象的文件占用的空间。如果变量的值无需通过序列化就可以确定，则不应对其进行序列化。

要将实例变量排除在序列化之外，可以使用限定符 `transient`。

将该限定符放在创建变量的语句中，它位于变量的类或数据类型之前。下面的语句创建了一个名为 `limit` 的暂态 (`transient`) 变量：

```
public transient int limit = 55;
```

### 16.1.4 检查对象的序列化字段

序列化对象时，需要考虑的一个重要因素是，恶意程序员篡改序列化格式的对象难易程度。在 Java 中，序列化对象的文件格式既不是加密的，也不特别复杂。

根据序列化对象重建原来的对象时，不能依赖于构造方法来确保其字段有允许的值。

相反，需要对从流中读取的对象进行检查，看它是否包含可接受的值，对象可能包含一个 `readObject(ObjectInputStream)` 方法。

这个方法引发 `IOException` 和 `ClassNotFoundException` 异常，其格式如下：

```
private void readObject(ObjectInputStream ois) {
    ois.defaultReadObject();
}
```

它是私有的。在这个方法中，对象流的 `defaultReadObject()` 方法将序列化字段读入到对象中，在对象中，可以对字段进行检查，以确保它们的值是可接受的。

如果不可接受，可引发 `IOException`，指出发生了与序列化相关的错误。

可在 `Message` 类中加入下述方法，以拒绝 `from` 值为空的序列化对象：

```
private void readObject(ObjectInputStream ois)
    throws IOException, ClassNotFoundException {

    ois.defaultReadObject();
    if (from.length() < 1) {
        throw new IOException("Null sender in message.");
    }
}
```

## 16.2 使用反射来检查类和方法

第3章中介绍了如何创建 `Class` 对象，它表示对象所属的类。在 Java 中，每个对象都继承了方法 `getClass()`，该方法指出对象所属的类或接口。下面的语句使用变量 `key` 引用的对象创建了一个名为 `keyclass` 的 `Class` 对象：

```
Class keyClass = key.getClass();
```

通过调用 `Class` 对象的 `getName()` 方法，可以获悉其类名：

```
String keyName = keyClass.getName();
```

这些特性是 Java 的反射特性。反射是一种这样的技术，即让 Java 类（如您编写的程序）能够获悉其他类的细节。

通过反射，Java 程序能够装载它一无所知的类，了解这个类的变量、方法和构造函数，并使用它们。

反射的用途之一是，在读取对象时判断对象所属的类。

### 16.2.1 检查和创建类

Class 类（位于 java.lang 中）用于了解和创建类、接口，甚至基本数据类型。

除使用 getClass() 外，您还可以在类、接口、数组或基本类型的名称后面加上 .class 来创建 Class 对象，如下述范例所示：

```
Class keyClass = KeyClass.class;
Class thr = Throwable.class;
Class floater = float.class;
Class floatArray = float[].class;
```

您还可以使用类方法 forName() 来创建 Class 对象，该方法接受一个参数：一个包含已有类名的字符串。下面的语句创建了一个表示 JLabel 的 Class 对象，JLabel 是 javax.swing 包中的一个类：

```
Class lab = Class.forName("javax.swing.JLabel");
```

如果指定的类找不到，forName() 方法将引发 ClassNotFoundException 异常，因此您必须在 try-catch 块中调用 forName() 方法或以其他方式处理它。

要获取包含 Class 对象表示的类名的字符串，可以调用该对象的 getName() 方法。对于类和接口，该名称由类名及其所属的包的名称组成；对于基本数据类型，则为相应类型的名称（如 int、float 或 double）。

对于表示数组的 Class 对象，调用其 getName() 方法时，处理方式略有不同。返回的名称以左方括号打头，左方括号的数目等于数组的维数：对于 float[]，以 [ 打头；对于 int[][]，以 [[ 打头；对于 KeyClass[][][]，以 [[[ 打头，以此类推。

如果数组为基本类型，则接下来的一部分为一个表示类型的字符，如表 16.1 所示。

表 16.1 基本类型的类型标识

字 符	基本类型
B	byte
C	char
D	double
F	float
I	int
J	long
S	short
Z	boolean

对于对象数组，则接下来为 L 和类名。例如，如果对 String[][] 数组调用 getName() 方法，则返回值为 [[Ljava.lang.String。

您还可以使用 Class 类来创建新对象。为此，可以调用 Class 对象的新实例 newInstance() 方法，并将它强制转换为正确的类。例如，如果有一个名为 thr 的 Class 对象，它表示 Throwable 接口，则可以这样来创建一个新的对象：

```
Throwable thr2 = (Throwable)thr.newInstance();
```

方法 `newInstance()` 可能引发下面几种异常。

- `IllegalAccessException`: 您无权访问这个类, 这可能是由于它不是公有的, 也可能是由于它属于另一个包。

- `InstantiationException`: 这个类是抽象的, 无法创建其对象。

- `SecurityViolation`: 您无权创建这个类的对象。

调用 `newInstance()` 且没有引发异常时, 将调用相应类的构造函数 (且不提供任何参数) 来创建新对象。

#### 注意

对于其构造函数需要参数的类, 不能采用这种方式来创建其对象, 而必须使用 `Constructor` 类的 `newInstance()` 方法来创建, 这将在本章后面进行介绍。

### 16.2.2 处理类的各个部分

虽然 `Class` 类位于 `java.lang` 包中, 但对反射提供主要支持的是 `java.lang.reflect` 包, 它包含以下类。

- `Field`: 管理和查找有关类和实例变量的信息。
- `Method`: 管理类方法和实例方法。
- `Constructor`: 管理构造函数——用于创建类实例的特殊方法。
- `Array`: 管理数组。
- `Modifier`: 对有关类、变量和方法的限定符信息进行解码。

这些反射类都有用于处理类中元素的方法。

`Method` 对象存储了有关类中某个方法的信息。要获悉类中的所有方法, 可创建这个类的 `Class` 对象, 并调用该对象的 `getDeclaredMethods()` 方法。该方法返回一个 `Method` 对象数组, 其中包含类中不是从超类那里继承的所有方法。如果没有这样的方法, 则该数组的长度将为 0。

`Method` 类有几个很有用的实例方法:

- `getParameterTypes()`: 返回一个 `Class` 对象数组, 其中包含方法特征标中的所有参数。
- `getReturnType()`: 返回一个 `Class` 对象, 它表示方法的返回类型, 这可能是类, 也可能是基本类型。
- `getModifiers()`: 返回一个 `int` 值, 指出用于方法的限定符, 如 `public`、`private` 等。

由于 `getParameterTypes()` 和 `getReturnType()` 方法返回 `Class` 对象, 所以您可以对这些对象调用 `getName()` 方法, 以获悉更详细的信息。

要使用 `getModifiers()` 返回的 `int` 值, 最简单的方式是用它作为参数来调用 `Modifier` 类的方法 `toString()`。例如, 如果有一个名为 `current` 的 `Method` 对象, 则可以使用下面的代码来显示其限定符:

```
int mods = current.getModifiers();
System.out.println(Modifier.toString(mods));
```

`Constructor` 类有几个与 `Method` 类相同的方法, 包括 `getModifiers()` 和 `getName()`。这里没有方法 `getReturnType()`, 这是因为构造函数没有返回值。

要获悉与 `Class` 对象相关联的所有构造函数, 可调用其 `getConstructors()` 方法。这将返回一个 `Constructor` 数组。

要获悉特定的构造函数, 首先创建一个 `Class` 对象数组, 它表示该构造函数接受的每个参数。然后, 调用 `getConstructors()`, 并将该 `Class` 数组作为参数传递给它。

例如, 如果有一个 `KeyClass(String, int)` 构造函数, 可以使用下述语句来创建一个表示该构造函数的 `Constructor` 对象:

```

Class kc = KeyClass.class;
Class[] cons = new Class[2];
cons[0] = String.class;
cons[1] = int.class;
Constructor c = kc.getConstructor(cons);

```

如果没有接受的参数与 Class[] 数组匹配的构造函数，则方法 `getConstructor(Class[])` 将引发 `NoSuchMethodException` 异常。

有了 `Constructor` 对象后，可以调用它的 `newInstance(Object[])` 方法，以使用该构造函数创建一个新实例。

### 16.2.3 检查类

为应用前面介绍的知识，程序清单 16.3 中的 Java 应用程序 `MethodInspector` 使用了反射来检测类中的方法。

程序清单 16.3 完整的 `MethodInspector.java` 源代码

```

1: import java.lang.reflect.*;
2:
3: public class MethodInspector {
4:     public static void main(String[] arguments) {
5:         Class inspect;
6:         try {
7:             if (arguments.length > 0)
8:                 inspect = Class.forName(arguments[0]);
9:             else
10:                inspect = Class.forName("MethodInspector");
11:            Method[] methods = inspect.getDeclaredMethods();
12:            for (int i = 0; i < methods.length; i++) {
13:                Method methVal = methods[i];
14:                Class returnVal = methVal.getReturnType();
15:                int mods = methVal.getModifiers();
16:                String modVal = Modifier.toString(mods);
17:                Class[] paramVal = methVal.getParameterTypes();
18:                StringBuffer params = new StringBuffer();
19:                for (int j = 0; j < paramVal.length; j++) {
20:                    if (j > 0)
21:                        params.append(", ");
22:                    params.append(paramVal[j].getName());
23:                }
24:                System.out.println("Method: " + methVal.getName() + "()");
25:                System.out.println("Modifiers: " + modVal);
26:                System.out.println("Return Type: " + returnVal.getName());
27:                System.out.println("Parameters: " + params + "\n");
28:            }
29:        } catch (ClassNotFoundException c) {
30:            System.out.println(c.toString());
31:        }
32:    }
33: }

```

应用程序 `MethodInspector` 显示了类中的公有方法的信息，这个类是您在命令行中指定的（如果没有指定，则为 `MethodInspector`）。要运行该程序，请在命令行上执行如下命令：

```
java MethodInspector java.util.Random
```

如果运行该应用程序时指定的是 `java.util.Random` 类，则输出如下（其中省略了一些方法）：

```

Method: writeObject()
Modifiers: private synchronized
Return Type: void
Parameters: java.io.ObjectOutputStream

Method: next()
Modifiers: protected
Return Type: int

```

```
Parameters: int
...
Method: setSeed()
Modifiers: public synchronized
Return Type: void
Parameters: long
```

通过使用反射，应用程序 `MethodInspector` 可以了解类中的每个方法。

该应用程序的第 7~10 行创建了一个 `Class` 对象。如果运行 `MethodInspector` 时，将类名指定为命令行参数，则调用方法 `Class.forName()` 时将以这个类名为参数；否则以 `MethodInspector` 为参数。

创建 `Class` 对象之后，第 11 行调用其 `getDeclaredMethods()` 方法，以找出这个类中的所有方法（从超类继承而来的方法除外）。这些方法将被存储在一个 `Method` 对象数组中。

第 12~28 行的 `for` 循环遍历类中的每个方法，存储其返回类型、限定符和参数，然后显示它们。

显示返回类型的语句很简单：第 14 行调用每一 `Method` 对象的 `getReturnType()` 方法，并将返回值存储在一个 `Class` 对象中，然后第 26 行显示该对象的名称。

第 15 行调用 `Method` 对象的 `getModifiers()` 方法，这将返回一个整数，指出用于该方法的所有限定符。第 16 行调用类方法 `Modifier.toString()`，并将该整数作为参数传递给它，这将返回所有限定符的名称。

第 19~23 行遍历 `Class` 对象数组，该数组中包含方法的所有参数。第 22 行将各个参数的名称加入到名为 `params` 的 `StringBuffer` 对象中。

反射最常见的用途是，被诸如类浏览器和调试器等工具用来获悉更详细的有关被浏览或调试的对象所属类的信息。

#### 注意

在 `JavaBeans` 中，也需使用反射。`JavaBeans` 是一种创建 `Java` 类的方法，使用这种方法创建的 `Java` 类可在编程环境中操纵。这些类被称为 `beans`，程序员可通过将 `beans` 载入界面、对其进行定制并控制其交互来创建 `Java` 应用程序。

## 16.3 总结

虽然 `Java` 一直是一种以网络为中心的语言，但本章介绍的主题表明，该语言在沿两个方向扩展。

对象序列化表明，使用 `Java` 创建的对象在 `Java` 程序终止后仍可存在。您可以将程序中创建的对象保存到诸如硬盘等存储设备中，并在程序终止后恢复它。

持久化是一种将程序元素保存供以后使用的高效方式。

## 16.4 问与答

问：应结合使用对象流和用于处理字符流的 `Writer` 类、`Reader` 类吗？

答：类 `ObjectInputStream` 和 `ObjectOutputStream` 独立于 `java.io` 包中的字节流和字符流超类，虽然它们包含很多与字节类类似的方法。

没有必要结合使用 `Writer`（或 `Reader`）类和对象流，因为可以使用对象流类及其超类（`InputStream` 和 `OutputStream`）来完成这样的任务。

问：对象被序列化时，其中的私有变量和对象也将被保存吗？

答：将被保存。正如本章讨论的，使用序列化将对象装载到程序中，不会调用任何构造方法。因此，未被声明为 `transient` 的变量和对象都将被保存，以防对象缺少完成其功能所必需的内容。

在某些情况下，保存私有变量和对象可能带来安全隐患，尤其当这种变量用于存储密码或其他敏感数据时。

## 16.5 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 16.5.1 问题

1. 如果对一个表示 `String[]` 数组的 `Class` 对象调用 `getName()` 方法，将返回什么？
  - a. `java.lang.String`;
  - b. `Ljava.lang.String`;
  - c. `java.lang.String`。
2. 什么是持久化？
  - a. 对象在创建它的程序停止运行后继续存在；
  - b. 一个类能够支持多个线程；
  - c. 一种错误处理方法。
3. 哪个 `Class` 方法使用包含类名的字符串创建一个新的 `Class` 对象？
  - a. `newInstance()`;
  - b. `forName()`;
  - c. `getName()`。

答案

1. b，方括号指出数组的维数，L 表明这是一个对象数组，后面的类名的含义是不言自明的。
2. a，持久化指的是将通过序列化将对象存储到磁盘或其他存储介质中，以便以后重建对象。
3. b，如果找不到这个类，将引发 `ClassNotFoundException` 异常。

### 16.5.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

对于下述代码：

```
public class ClassType {
    public static void main(String[] arguments) {
        Class c = String.class;
        try {
            Object o = c.newInstance();
            if (o instanceof String)
                System.out.println("True");
            else
                System.out.println("False");
        } catch (Exception e) {
            System.out.println("Error");
        }
    }
}
```

该应用程序的输出是什么？

- a. `True`;
- b. `False`;



- c. `Error`;
- d. 该程序将无法通过编译。

答案 a

## 16.6 练习

为巩固本章介绍的知识，请尝试完成下面的练习：

1. 使用反射来编写一个 Java 程序，它接收一个类名作为命令行参数，并检查它是否是应用程序：所有应用程序都有 `main()` 方法，该方法使用限定符 `public static`，返回类型为 `void`，并接受一个 `String[]` 参数。
2. 编写一个程序，它使用 `Class` 对象来创建一个新对象，并使用 `newInstance()` 方法将该对象序列化到磁盘中。



## 第 17 章

# 通过 Internet 进行通信

Java 最初是作为一种控制家电设备网络的语言而被开发的。当初设计该语言时，主旨之一是连接机器，而现在仍是如此。

Java.net 包让 Java 程序能够通过网络进行通信。这个包为简单的网络操作提供了跨平台抽象，包括使用常见的 Web 协议建立连接和传输文件以及创建套接字。

结合使用输入/输出流，通过网络读写文件几乎与读写本地磁盘文件一样容易。

java.nio 包是对 Java 输入/输出类的扩展。

在本章中，您将编写网络 Java 程序：创建通过万维网加载文档、模仿流行的 Internet 服务并为客户提供信息的应用程序。

### 17.1 Java 的联网技术

联网技术指的是不同的计算机彼此连接并交换信息的能力。在 Java 中，基本的联网技术由 java.net 包中的类支持，这包括支持通过超文本传输协议 (HTTP) 和文件传输协议 (FTP) 进行连接和检索文件以及在底层处理基本的 UNIX 式套接字。

要同网络上的系统进行通信，可以采取 3 种简单的方式：

- 在小程序中使用统一资源定位符 (URL) 加载网页和其他资源；
- 使用套接字类 Socket 和 ServerSocket，它们建立到主机的标准套接字连接，并通过这种连接执行读写；
- 调用 `getInputStream()`，该方法建立到 URL 的连接，并通过该连接来获取数据。

#### 17.1.1 打开跨越网络的流

正如第 15 章介绍的，通过流将信息输入到 Java 程序中的方式有多种。选择使用哪些类和方法取决于信息的格式以及您要如何操纵它。

在 Java 程序中，可以访问的资源之一是万维网上的文本文档，这可能是超文本标记语言 (HTML) 文件、可扩展标记语言 (XML) 文件或其他类型的纯文本文档。

要加载网上的文本文档并逐行读取其中的内容，可以通过 4 步来实现：

1. 创建一个表示资源的网络地址的 URL 对象；
2. 创建一个 `URLConnection` 对象，它能够加载 URL 并连接到相应的站点；
3. 使用 `URLConnection` 对象的 `getContent()` 方法来创建一个 `InputStreamReader`，用于读取来自 URL 的数据流；
4. 使用输入流阅读器来创建一个 `BufferedReader` 对象，后者能够高效地从输入流中读取字符。



Web 文档和 Java 程序之间将进行大量的交互。URL 用于建立 URL 连接，后者用于建立输入流阅读器，而输入流阅读器用于建立缓冲输入流阅读器。由于需要对可能发生的异常进行捕获，这增加了复杂程度。

要加载资源，首先必须创建 URL 类的一个实例，它表示要加载的资源地址。URL 是 uniform resource locator（统一资源定位器）的缩写，它是可以通过 Internet 进行访问的文件或其他资源的唯一地址。

URL 类位于 java.net 包中，因此您必须在程序导入这个包或使用全名来引用这个类。

要创建新的 URL 对象，可使用下述 4 个构造函数之一。

- URL(String): 使用完整的网络地址来创建一个 URL 对象。
- URL(URL, String): 将指定的 URL 作为基本地址，指定的 String 作为相对路径来创建一个 URL 对象。
- URL(String, String, int, String): 根据协议（如 http 或 ftp）、主机名（如 www.cnn.com 或 web.archive.org）、端口号（对于 HTTP 为 80）和文件名或路径名创建一个新的 URL 对象。
- URL(String, String, String): 与前一个构造函数相同，只是没有端口号。

使用构造函数 URL(String)时，必须处理 MalformedURLException 异常，该异常在指定的 String 不是有效的 URL 时引发。可以在 try-catch 块处理这些异常：

```
try {
    URL load = new URL("http://www.sampublishing.com");
} catch (MalformedURLException e) {
    System.out.println("Bad URL");
}
```

程序清单 17.1 中的应用程序 WebReader 通过前面介绍的 4 个步骤连接到一个网站，并读取其中的一个文本文档。加载该文档后，将其内容显示到一个文本区域中。

**程序清单 17.1 完整的 WebReader.java 源代码**

```
1: import javax.swing.*;
2: import java.awt.*;
3: import java.awt.event.*;
4: import java.net.*;
5: import java.io.*;
6:
7: public class WebReader extends JFrame {
8:     JTextArea box = new JTextArea("Getting data ...");
9:
10:    public WebReader() {
11:        super("Get File Application");
12:        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13:        setSize(600, 300);
14:        JScrollPane pane = new JScrollPane(box);
15:        add(pane);
16:        setVisible(true);
17:    }
18:
19:    void getData(String address) throws MalformedURLException {
20:        setTitle(address);
21:        URL page = new URL(address);
22:        StringBuffer text = new StringBuffer();
23:        try {
24:            HttpURLConnection conn = (HttpURLConnection)
25:                page.openConnection();
26:            conn.connect();
27:            InputStreamReader in = new InputStreamReader(
28:                (InputStream) conn.getContent());
29:            BufferedReader buff = new BufferedReader(in);
30:            box.setText("Getting data ...");
31:            String line;
32:            do {
```



```

33:         line = buff.readLine();
34:         text.append(line + "\n");
35:     } while (line != null);
36:     box.setText(text.toString());
37: } catch (IOException ioe) {
38:     System.out.println("IO Error:" + ioe.getMessage());
39: }
40: }
41:
42: public static void main(String[] arguments) {
43:     if (arguments.length < 1) {
44:         System.out.println("Usage: java WebReader url");
45:         System.exit(1);
46:     }
47:     try {
48:         WebReader app = new WebReader();
49:         app.getData(arguments[0]);
50:     } catch (MalformedURLException mue) {
51:         System.out.println("Bad URL: " + arguments[0]);
52:     }
53: }
54: }

```

要运行应用程序 WebReader，可指定一个命令行参数：一个 URL。例如：  
 java WebReader http://www.rssboard.org/rss-feed

可使用任何 URL。前面的例子加载了 RSS 文件中的一个页面，如图 17.1 所示。

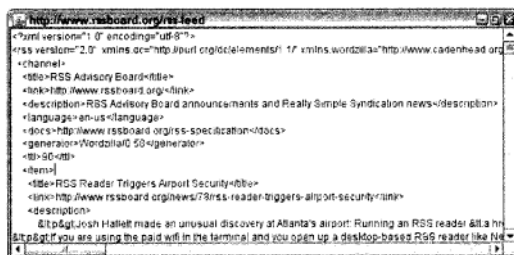


图 17.1 运行应用程序 WebReader

WebReader 类中 2/3 的代码用于运行应用程序、创建用户界面和一个有效的 URL 对象。getData() 方法加载 Web 文档中的数据并将其显示在一个文本区域中。

首先创建了 3 个对象：URLConnection、InputStreamReader 和 BufferedReader。这些对象将被用来将数据从 Internet 上读取到 Java 程序中。另外，还创建了两个对象：一个 String 和一个 StringBuffer，用于存储取回的数据。

第 24~26 行建立一个 HTTP URL 连接，这对于获得跨越网络的输入流是必不可少的。

第 27~28 行使用连接的 getContent() 方法来创建一个输入流阅读器。该方法返回一个输入流，后者表示到 URL 的连接。

第 29 行使用输入流阅读器创建一个缓冲输入流阅读器——一个名为 buff 的 BufferedReader 对象。

有了缓冲阅读器后，便可以调用其 readLine() 方法来从输入流中读取一行文本。该缓冲阅读器将字符放到缓冲区中，并在被请求时将它们从缓冲区中取出。

第 32~35 行的 while 循环逐行读取 Web 文档的内容，并将其追加到用于存储页面文本的 StringBuffer 对象中。

读取所有的数据后，第 36 行调用方法 toString() 将字符串缓冲区转换为字符串，然后调用文本区

域组件的 `append(String)` 方法将转换结果放到文本区域中。

`URLConnection` 类有多个影响 HTTP 请求或提供额外信息的方法。

- `getHeaderField(int)`: 返回一个字符串, 其中包含诸如 `Server` (存储文档的 Web 服务器) 或 `Last-Modified` (文档最后一次修改的日期) 等 HTTP 报头。

报头从 0 开始编号。达到报头末尾时, 该方法返回 `null`。

- `getHeaderFieldKey(int)`: 返回一个字符串, 其中包含指定报头的名称 (如 `Server` 或 `Last-Modified`), 或者返回 `null`。

- `getResponseCode()`: 返回一个整数, 指出请求的 HTTP 响应编码, 如 200 (有效请求) 或 404 (请求的文档找不到)。

- `getResponseMessage()`: 返回一个字符串, 其中包含 HTTP 响应编码和解释性消息, 如 “HTTP/1.0 200 OK”。

对于每个有效的响应编码, `URLConnection` 类中都有一个相应的整型类变量, 如 `HTTP_OK`、`HTTP_NOT_FOUND` 和 `HTTP_MOVED_PERM`。

- `getContentType()`: 返回一个字符串, 其中包含 Web 文档的 MIME 类型。有些可能的类型包括 `text/html` (网页) 和 `text/xml` (XML 文件)。

- `SetFollowRedirects(boolean)`: 决定是遵循 (`true`) URL 重定向请求还是忽略 (`false`)。支持重定向请求时, Web 服务器可以将 URL 请求转到正确的地址。

可以将下述代码加入到在应用程序 `WebReader` 的 `getData()` 方法中, 以便同时显示报头和文档中的文本:

```
String key;
String header;
int i = 0;
do {
    key = conn.getHeaderFieldKey(i);
    header = conn.getHeaderField(i);
    if (key == null) {
        key = "";
    } else {
        key = key + ": ";
    }
    if (header != null) {
        text.append(key + header + "\n");
    }
    i++;
} while (header != null);
text.append("\n");
```

### 17.1.2 套接字

对于 URL 和 `URLConnection` 类的功能不能满足其要求的网络应用程序 (例如, 使用其他协议或更通用的网络应用程序), Java 提供了 `Socket` 和 `ServerSocket` 类, 它们是标准的传输控制协议 (TCP) 套接字编程技术的抽象。

`Socket` 类提供了一个类似于标准 UNIX 套接字的客户端套接字接口。要建立连接, 可创建一个 `Socket` 实例 (其中 `hostName` 是要连接的主机, `portNumber` 是端口号):

```
Socket connection = new Socket(hostName, portNumber);
```

创建套接字后, 应设置其超时值, 这个值决定了应用程序将为数据的到来等待多长时间。为此, 可调用套接字的 `setSoTimeout(int)` 方法, 该方法接受一个参数: 等待时间 (单位为毫秒):

```
connection.setSoTimeout(50000);
```

调用上述方法后, 从 `connection` 表示的套接字中读取数据时, 将只等待 50 000ms (即 50s)。如果超时, 将引发 `InterruptedIOException` 异常, 因此您可以在 `try-catch` 块关闭套接字或再次读取数据。

如果使用套接字的程序没有设置超时，它可能一直等待下去，直到数据到来。

#### 提示

为避免这种问题，通常将网络操作放在单独的线程中，并让它和程序的其他部分分开运行，第 7 章中将这种技术用于动画。

建立套接字后，可以使用输入/输出流来读写它：

```
BufferedInputStream bis = new
    BufferedInputStream(connection.getInputStream());
DataInputStream in = new DataInputStream(bis);
```

```
BufferedOutputStream bos = new
    BufferedOutputStream(connection.getOutputStream());
DataOutputStream out = new DataOutputStream(bos);
```

实际上，并不需要给这些对象命名，因为它们只用于创建流或流阅读器。为提高效率，可使用一个名为 `sock` 的 `Socket` 对象来将上述语句合并为一条：

```
DataInputStream in = new DataInputStream(
    new BufferedInputStream(
        sock.getInputStream()));
```

在这条语句中，`sock.getInputStream()` 调用返回一个与套接字相关联的输入流；然后使用该输入流创建了 `BufferedInputStream`，而后者被用来创建一个 `DataInputStream`。这样，只使用了变量 `sock` 和 `in`，从连接中接收数据然后关闭该连接时需要使用这两个变量。中间对象（`BufferedInputStream` 和 `InputStream`）只被使用一次。

使用完套接字后，别忘了调用 `close()` 方法来关闭它。这也将关闭您为套接字建立的所有输入/输出流。例如：

```
connection.close();
```

套接字编程可用于使用 TCP/IP 联网技术提供的众多服务中，包括 `telnet`、简单邮件协议（SMTP，用于接收邮件）、网络传输新闻协议（NNTP，用于 Usenet 新闻）和 `finger`。

其中 `finger` 是向系统查询其用户的协议。通过建立 `finger` 服务器，系统管理员让连接到 Internet 的机器能够回答有关用户信息的查询。用户可以通过创建 `plan` 文件来提供有关自己的信息，这些信息将被发送给使用 `finger` 来询问的用户。

虽然由于安全方面的原因，近几年来 `finger` 已不再被使用，但在博客出现前，它是 Internet 用户发布有关其情况和行为的最为流行方式。您可以对另一所大学的朋友的账号使用 `finger`，以查看他是否在线，并读取其最新的 `plan` 文件。

#### 注意

当前，仍有人通过 `finger` 写日志，它们是游戏编程社区。Blues News 网站包含到多个这种社区的链接，其网址为 <http://www.bluesnews.com/plans/476/>。

作为一个套接字编程练习，应用程序 `Finger` 是一个基本的 `finger` 客户程序（见程序清单 17.2）。

#### 程序清单 17.2 完整的 `Finger.java` 源代码

```
1: import java.io.*;
2: import java.net.*;
3: import java.util.*;
4:
5: public class Finger {
6:     public static void main(String[] arguments) {
7:         String user;
8:         String host;
9:         if ((arguments.length == 1) && (arguments[0].indexOf("@") > -1)) {
10:             StringTokenizer split = new StringTokenizer(arguments[0],
11:                 "@");
12:             user = split.nextToken();
13:             host = split.nextToken();
14:         } else {
```

```

15:         System.out.println("Usage: java Finger user@host");
16:         return;
17:     }
18:     try {
19:         Socket digit = new Socket(host, 79);
20:         digit.setSoTimeout(20000);
21:         PrintStream out = new PrintStream(digit.getOutputStream());
22:         out.print(user + "\015\012");
23:         BufferedReader in = new BufferedReader(
24:             new InputStreamReader(digit.getInputStream()));
25:         boolean eof = false;
26:         while (!eof) {
27:             String line = in.readLine();
28:             if (line != null)
29:                 System.out.println(line);
30:             else
31:                 eof = true;
32:         }
33:         digit.close();
34:     } catch (IOException e) {
35:         System.out.println("IO Error: " + e.getMessage());
36:     }
37: }
38: }

```

发出 finger 请求时, 指定用户名, 并加上@和主机名, 其格式与电子邮件地址相同。一个真实的例子是 johnc@idsoftware.com, 这是 id Software 的创始人 John Carmack 的 finger 地址。您可以这样运行应用程序 Finger 来请求他的.plan 文件:

```
java Finger johnc@idsoftware.com
```

如果 johnc 在 finger 服务器 idsoftware.com 上有账号, 则该程序的输出将是他的.plan 文件和其他信息 (虽然看起来他已不再更新这些信息)。如果用户找不到, 服务器将指出这一点。

Blues News 网站包含其他提供.plan 更新的游戏设计者的地址, 如 id Software 的开发人员 Timothee Besset (ttimo@idsoftware.com)。

应用程序 Finger 使用类 StringTokenizer 来将一个格式为 user@host 的地址转换为两个 String 对象: user 和 host (第 10~13 行)。

程序执行了下述套接字操作。

- 第 19~20 行: 使用主机名和端口 79 (通常保留给 finger 服务) 创建了一个新的 Socket, 并将超时值设置为 20s。
- 第 21 行: 使用该套接字来获得一个 OutputStream, 后者被用来创建一个 PrintStream 对象。
- 第 22 行: finger 协议要求通过套接字来发送用户名, 并在后面加上回车 ('\015') 和换行符 ('\012')。这是通过调用 PrintStream 对象的 print() 方法来完成的。
- 第 23~24 行: 发送用户名后, 必须在套接字上创建一个输入流来接收来自 finger 服务器的输入。通过将几个创建流的表达式组合在一起, 创建了 BufferedReader 流 in。这种流适合读取 finger 输入, 因为它每次读取一行文本。
- 第 26~32 行: 通过循环不断地从缓冲阅读器中读取文本行, 到达服务器的输出的末尾后, in.readLine() 将返回 null, 因此循环结束。

用于通过套接字同 finger 服务器进行通信的技术也可用于连接到其他流行的 Internet 服务。只需修改第 19 行的端口号并做其他一些修改, 该程序便变成了一个 telnet 或 Web 读取客户程序。

### 17.1.3 Socket 服务器

服务器端套接字的工作原理与客户端套接字类似, 只是它还包含 accept() 方法。服务器套接字监听 TCP 端口上的客户连接; 当客户连接到该端口时, accept() 方法将接收该连接。通过使用客户套接字

和服务套接字，可以创建通过网络进行通信的应用程序。

要创建服务器套接字，并将其绑定到某个端口，可创建一个 `ServerSocket` 实例，并将该端口号作为参数传递给构造函数，如下例所示：

```
ServerSocket servo = new ServerSocket(8888);
```

然后，使用方法 `accept()` 来监听该端口（并接收来自客户的连接）：

```
servo.accept();
```

建立套接字连接后，可以分别使用输入流和输出流来从客户端读取数据和将数据写入客户端。

要扩展套接字类的行为——如允许网络连接跨越防火墙或代理，可以使用抽象类 `SocketImpl` 和接口 `SocketImplFactory` 来创建一个新的传输层套接字实现。

这种设计符合 Java 套接字类的初衷：让这些类能够被移植到使用不同传输机制的系统中。这种机制存在的问题是，虽然在简单情况下管用，但它不允许您将其他协议添加到 TCP 上（例如，以实现诸如安全套接字层[SSL]等加密机制），也不允许单个 Java 运行环境包含多个套接字实现。

因此，在 Java 1.0 后，套接字被扩展了，使 `Socket` 和 `ServerSocket` 类不再是 `final` 和不可扩展的。您可以创建这些类的子类，并使用默认的套接字实现或自己的实现。这使得网络功能灵活得多。

#### 17.1.4 设计服务器应用程序

下面的 Java 程序示例使用 `Socket` 类来实现一个简单的、基于网络的服务器应用程序。

应用程序 `TimeServer` 与连接到端口 4413 的客户建立连接、显示当前时间，然后关闭该连接。

应用程序要充当服务器，必须至少监听主机上一个端口的客户连接。这里监听的是端口 4415，但也可以监听 1 024~65 535 之间的任何一个端口。

##### 注意

端口 0 ~ 1 023 的用途由 Internet 地址分配机构控制，但编号更大的端口也可能已被占用，只是没有正式通知而已。为自己的客户 / 服务器应用程序选择端口号时，应了解哪些端口已被他人使用了。

检测到客户后，服务器将创建一个表示当前日期和时间的 `Date` 对象，并将其作为 `String` 发送给客户。

服务器和客户之间交换信息时，几乎所有的工作都是由服务器完成的。客户只是负责建立到服务器的连接，并显示从服务器那里收到的信息。

虽然您可以开发一个简单的客户程序，但也可以将任何 `telnet` 应用程序用作客户程序，只要它能够连接到指定的端口。Windows 包含了一个名为 `telnet` 的命令行应用程序，您可以将它用作客户程序。

程序清单 17.3 列出了服务器应用程序的源代码。

##### 程序清单 17.3 完整的 `TimeServer.java` 源代码

```
1: import java.io.*;
2: import java.net.*;
3: import java.util.*;
4:
5: public class TimeServer extends Thread {
6:     private ServerSocket sock;
7:
8:     public TimeServer() {
9:         super();
10:        try {
11:            sock = new ServerSocket(4415);
12:            System.out.println("TimeServer running ...");
```

```

13:         } catch (IOException e) {
14:             System.out.println("Error: couldn't create socket.");
15:             System.exit(1);
16:         }
17:     }
18:
19:     public void run() {
20:         Socket client = null;
21:
22:         while (true) {
23:             if (sock == null)
24:                 return;
25:             try {
26:                 client = sock.accept();
27:                 BufferedOutputStream bos = new BufferedOutputStream(
28:                     client.getOutputStream());
29:                 PrintWriter os = new PrintWriter(bos, false);
30:                 String outLine;
31:
32:                 Date now = new Date();
33:                 os.println(now);
34:                 os.flush();
35:
36:                 os.close();
37:                 client.close();
38:             } catch (IOException e) {
39:                 System.out.println("Error: couldn't connect to client.");
40:                 System.exit(1);
41:             }
42:         }
43:     }
44:
45:     public static void main(String[] arguments) {
46:         TimeServer server = new TimeServer();
47:         server.start();
48:     }
49:
50: }

```

应用程序 TimeServer 在端口 4415 上创建一个服务器套接字。当客户端连接时,将根据缓存输出流创建一个 PrintWriter 对象,以便能够将当前时间发送给客户端。

发送该字符串后,写入器的方法 flush() 和 close() 结束数据交换并关闭套接字,以便等待新连接。

### 17.1.5 测试服务器

仅当应用程序 TimeServer 正运行时,客户程序才能连接到它。因此首先必须运行该服务器:

```
java TimeServer
```

如果成功运行,服务器将只显示一行输出:

```
TimeServer running ...
```

运行服务器后,您可以在计算机上使用 telnet 程序通过端口 4415 连接到它。

要在 Windows 中运行 telnet:

- 在 Windows 95、98、Me、NT 或 2000 中,单击“开始”按钮并选择“运行”以打开“运行”对话框,然后在“打开”文本框中输入 telnet 并按回车键。这将打开一个 telnet 窗口。

要建立 telnet 连接,请选择菜单“连接”>“远程系统”打开“连接”对话框。在“主机名”文本框中输入 localhost,在“端口”文本框中输入 4415,并保留“终端类型”文本框中的默认值: vt100。

- 在 Windows XP/2003 中,单击“开始”按钮并选择“运行”以打开“运行”对话框,然后在“打开”文本框中输入 telnet localhost 65156,并按回车键。

- 主机名 localhost 表示您的机器——运行该服务器应用程序的系统。您可以在本地对服务器应用程序进行测试,然后再将其部署到 Internet 上。

要在 telnet 客户和应用程序 TimeServer 之间建立套接字连接，您可能需要登录 Internet，这取决于您系统的 Internet 连接配置。

如果该服务器位于与 Internet 相连的另一台计算机上，则需要指定该计算机的主机名或 IP 地址，而不是 localhost。

使用 telnet 连接到 TimeServer 应用程序后，它将显示服务器的当前时间，然后关闭连接。telnet 程序的输出与下面类似：

```
Thu Feb 15 22:13:58 EST 2007
Connection to host lost.
Press any key to continue...
```

## 17.2 java.nio 包

java.nio 包中的类扩展了 Java 语言的网络功能，对读写数据，使用文件、套接字和内存以及处理文本很有帮助。

当您使用新的输入/输出特性时，常常需要用到另外两个相关的包：java.nio.channels 和 java.nio.charset。

### 17.2.1 缓冲区

java.nio 包提供了对缓冲区 (buffer) 的支持，缓冲区是一种对象，表示存储在内存中的数据流。

缓冲区常被用来提高那些读取输入和发送输出的程序的性能。它们让程序能够将大量的数据存储在内存中，这样读写或修改这些数据时速度将快得多。

对于 Java 中的每种基本数据类型，都有相应的缓冲类：

- ByteBuffer,
- CharBuffer,
- DoubleBuffer,
- FloatBuffer,
- IntBuffer,
- LongBuffer,
- ShortBuffer.

上述缓冲类都有一个名为 wrap() 的静态方法，可用于创建缓冲区。该方法只接受一个参数：一个相应数据类型的数组。

例如，下面的语句创建了一个 int 数组和一个 IntBuffer，后者将这些整数存储在作为缓冲区的内存中：

```
int[] temperatures = { 90, 85, 87, 78, 80, 75, 70, 79, 85, 92, 99 };
IntBuffer tempBuffer = IntBuffer.wrap(temperatures);
```

缓冲区存储下一个将被读写的数据项的位置，以跟踪自己是如何被使用的。创建缓冲区后，可以调用其 get() 方法来读取当前位置的数据项。下面的语句显示前面创建的整数缓冲区中的所有内容：

```
for (int i = 0; tempBuffer.remaining() > 0; i++)
    System.out.println(tempBuffer.get());
```

另一种创建缓冲区的方式是，首先建立一个空的缓冲区，然后加入数据。要创建缓冲区，可调用缓冲区类的静态方法 allocate(int)，并将缓冲区的大小作为参数传递给它。

有 5 个 put() 方法可用于将数据存储在缓冲区中（或替换缓冲区中的数据）。这些方法接受的参数取决于被操纵的缓冲区的类型。对于整数缓冲区：

- put(int)：将指定的整数存储在缓冲区的当前位置，然后将位置加 1。



- `put(int, int)`: 将指定的整数 (第二个参数) 存储到缓冲区的指定位置 (第一个参数)。
- `put(int[])`: 将指定的整数数组中的所有元素存储到缓冲区中——从缓冲区的第一个位置开始。
- `put(int[], int, int)`: 将指定数组中的所有或部分元素存储到缓冲区中。其中第二个参数指定了第一个元素将被存储在缓冲区的什么位置, 而第三个参数指定了要将数组中的多少个元素存储到缓冲区中。
- `put(IntBuffer)`: 将指定的整数缓冲区存储到当前整数缓冲区中——从当前缓冲区的第一个位置开始。

将数据存储在缓冲区中时, 必须跟踪当前位置, 以便知道接下来的数据将被存储到什么位置。

要知道当前位置, 可调用缓冲区的 `position()` 方法。该方法返回一个整数, 指出当前位置。如果为 0, 则说明位于缓冲区的开头。

要改变当前位置, 可调用 `position(int)` 方法, 并将目标位置作为参数传递给它。

使用缓冲区时, 需要跟踪的另一个重要的位置是缓冲区中最后一个数据项的位置。

如果缓冲区总是满的, 则没有必要跟踪最后一个数据项的位置。在这种情况下, 缓冲区的最后一个位置肯定存储了数据。

然而, 如果缓冲区存储的数据量少于分配给它的内存空间, 则将数据读入到缓冲区后, 应调用缓冲区的 `flip()` 方法。这将当前位置设置为前一次读入的数据的开头, 并将最后一个数据项的位置设置为前一次读入的数据的末尾。

本章后面将使用 `byte` 缓冲区来存储从 Internet 上的网页中加载的数据。在这种情况下, 必须调用 `flip()`, 因为请求时您并不知道页面中包含多少数据。

如果缓冲区的大小为 1024 字节, 而页面中包含的数据量为 1500 字节, 则首次读取数据时将把 1024 字节的数据加入到缓冲区中, 将其填满。

第二次读取数据时, 将填充缓冲区的前 476 字节, 其余的为空。如果之后调用 `flip()`, 则当前位置将被设置为缓冲区的开头, 最后一项数据的位置为 476。

下面的代码创建一个 `int` 数据, 其中包含的是华氏温度。然后将华氏温度转换为摄氏温度, 并将结果存储到一个缓冲区中:

```
int[] temps = { 90, 85, 87, 78, 80, 75, 70, 79, 85, 92, 99 };
IntBuffer tempBuffer = IntBuffer.allocate(temperatures.length);
for (int i = 0; i < temps.length; i++) {
    float celsius = ( (float)temps[i] - 32 ) / 9 * 5;
    tempBuffer.put( (int)celsius );
};
tempBuffer.position(0);
for (int i = 0; tempBuffer.remaining() > 0; i++)
    System.out.println(tempBuffer.get());
```

将当前位置设置为缓冲区的开头后, 显示缓冲区的内容。

### byte 缓冲区

对于 `byte` 缓冲区, 您可以使用前面介绍的方法, 但它还包含其他方法。

`byte` 缓冲区包含用于存储和检索非 `byte` 数据的方法。

- `putchar(char)`: 将两个 `byte` 值存储到缓冲区中, 这两个 `byte` 值表示指定的 `char` 值。
- `putDouble(double)`: 将 8 个 `byte` 值存储到缓冲区中, 这 8 个 `byte` 值表示指定的 `double` 值。
- `putFloat(float)`: 将 4 个 `byte` 值存储到缓冲区中, 这 4 个 `byte` 值表示指定的 `float` 值。
- `putInt(int)`: 将 4 个 `byte` 值存储到缓冲区中, 这 4 个 `byte` 值表示指定的 `int` 值。
- `putLong(long)`: 将 8 个 `byte` 值存储到缓冲区中, 这 8 个 `byte` 值表示指定的 `long` 值。
- `putShort(short)`: 将 2 个 `byte` 值存储到缓冲区中, 这 2 个 `byte` 值表示指定的 `short` 值。

这些方法都将多个 `byte` 值存储到缓冲区中, 并将当前位置向前移动相应数目的字节。

另外，还有用于从 byte 缓冲区中获取非 byte 数据的方法：getChar( )、getDouble( )、getFloat( )、getInt( )、getLong( )和 getShort( )。

### 17.2.2 字符集

字符集位于 java.nio.charset 包中，这些类用于在 byte 缓冲区和字符缓冲区之间转换数据。

主要的类有 3 个。

- Charset：一个 Unicode 字符集，其中每个字符都有不同的 byte 值。
- Decoder：将一系列的 byte 值转换为一系列的字符。
- Encoder：将一系列的字符转换为一系列的 byte 值。

在 byte 缓冲区和字符缓冲区之间进行转换之前，必须首先创建一个 Charset 对象，它将字符映射到相应的 byte 值。

要创建字符集，可调用 Charset 类的静态方法 forName(String)，并将字符编码技术的名称作为参数传递给它。

Java 支持 6 种字符编码。

- US-ASCII：包含 128 个字符的 ASCII 字符集，是 Unicode 的基本拉丁字符，也叫 ISO646-US。
- ISO-8859-1：256 个字符的拉丁字母字符集，也叫 ISO-LATIN-1。
- UTF-8：包括 US-ASCII 和通用字符集（也叫 Unicode）的字符集，由全世界各种语言中的数千个字符组成。

- UTF-16BE：使用 16 位表示的通用字符集，其中的字节按 big-endian 顺序排列。
- UTF-16LE：使用 16 位表示的通用字符集，其中的字节按 little-endian 顺序排列。
- UTF-16：使用 16 位表示的通用字符集，使用可选择字节顺序标记来指出字节的排列顺序。

下面的语句创建了一个 ISO-8859-1 字符集的 Charset 对象：

```
Charset isoset = Charset.forName("ISO-8859-1");
```

有了字符集对象后，便可以使用它来创建编码器和解码器。要创建 CharsetDecoder 和 CharEncoder，可分别调用字符集对象的方法 newDecoder( )和 newEncoder( )。

要将 byte 缓冲区转换为字符缓冲区，可调用解码器的 decode(ByteBuffer)方法。该方法返回一个 CharBuffer，其中包含转换得到的字符。

要将字符缓冲区转换为 byte 缓冲区，可调用编码器的 encode(CharBuffer)方法。该方法返回一个 ByteBuffer，其中包含转换得到的 byte 值。

下面的语句使用 ISO-8859-1 字符集将一个名为 netBuffer 的 byte 缓冲区转换为一个字符缓冲区：

```
Charset set = Charset.forName("ISO-8859-1");
CharsetDecoder decoder = set.newDecoder();
netBuffer.position(0);
CharBuffer netText = decoder.decode(netBuffer);
```

#### 警告

使用解码器来创建字符缓冲区之前，方法调用 position(0)将 netBuffer 的当前位置重置到缓冲区的开头。刚开始使用缓冲区时，很容易忽略这一点，导致缓冲区中的数据比期望的少得多。

### 17.2.3 通道

缓冲区常与输入/输出流关联起来。您可以使用输入流中的数据来填充缓冲区或将缓冲区中的数据写入到输出流中。

为此,必须使用通道(channel)——一种将缓冲区和流连接起来的对象。通道类位于 java.nio.channels 包中。

可通过调用 getChannel() 方法来将通道与流关联起来, java.io 包中的一些类拥有这种方法。

FileInputStream 和 FileOutputStream 类拥有 getChannel() 方法, 它返回一个 FileChannel 对象。文件通道可用于读写和修改文件中的数据。

下面的语句创建了一个文件输入流和一个与该文件相关联的通道:

```
try {
    String source = "prices.dat";
    FileInputStream inSource = new FileInputStream(source);
    FileChannel inChannel = inSource.getChannel();
} catch (FileNotFoundException fne) {
    System.out.println(fne.getMessage());
}
```

创建文件通道后, 可以调用其 size() 方法来获悉文件中包含的字节数。如果要创建一个 byte 缓冲区来存储文件的内容, 则必须这样做。

要将通道中的字节读入到 ByteBuffer 中, 可使用方法 read(ByteBuffer, long)。其中第一个参数是缓冲区, 第二个参数是缓冲区的当前位置, 它决定了文件的内容将被存储到什么位置。

下面的语句扩展了前一个示例, 使用文件通道 inChannel 将文件读入到 byte 缓冲区中:

```
long inSize = inChannel.size();
ByteBuffer data = ByteBuffer.allocate( (int)inSize );
inChannel.read(data, 0);
data.position(0);
for (int i = 0; data.remaining() > 0; i++)
    System.out.print(data.get() + " ");
```

如果读取通道时发生问题, 将引发 IOException 错误。虽然上述 byte 缓冲区的大小与文件相同, 但并不一定非得这样。如果将文件读入到缓冲区旨在能够修改文件, 则可以分配一个比文件更大的缓冲区。

接下来的应用程序使用了前面介绍的新的输入/输出特性: 缓冲区、字符集和通道。

应用程序 BufferConverter 将一个小型文件的内容读入到 byte 缓冲区中, 显示缓冲区的内容, 将该缓冲区转换为字符缓冲区, 然后显示字符缓冲区的内容。

请输入程序清单 17.4 中的代码, 然后将其保存为 BufferConverter.java。

#### 程序清单 17.4 完整的 BufferConverter.java 源代码

```
1: import java.nio.*;
2: import java.nio.channels.*;
3: import java.nio.charset.*;
4: import java.io.*;
5:
6: public class BufferConverter {
7:     public static void main(String[] arguments) {
8:         try {
9:             // read byte data into a byte buffer
10:            String data = "friends.dat";
11:            FileInputStream inData = new FileInputStream(data);
12:            FileChannel inChannel = inData.getChannel();
13:            long inSize = inChannel.size();
14:            ByteBuffer source = ByteBuffer.allocate( (int)inSize );
15:            inChannel.read(source, 0);
16:            source.position(0);
17:            System.out.println("Original byte data:");
18:            for (int i = 0; source.remaining() > 0; i++)
19:                System.out.print(source.get() + " ");
20:
21:            // convert byte data into character data
22:            source.position(0);
23:            Charset ascii = Charset.forName("US-ASCII");
24:            CharsetDecoder toAscii = ascii.newDecoder();
25:            CharBuffer destination = toAscii.decode(source);
```



```

26:         destination.position(0);
27:         System.out.println("\n\nNew character data:");
28:         for (int i = 0; destination.remaining() > 0; i++)
29:             System.out.print(destination.get());
30:     } catch (FileNotFoundException fne) {
31:         System.out.println(fne.getMessage());
32:     } catch (IOException ioe) {
33:         System.out.println(ioe.getMessage());
34:     }
35: }
36: }

```

编译上述文件后，您需要一个 friends.dat 的拷贝——应用程序使用的小型字节数据文件。

#### 提示

您也可以自己创建文件。为此，打开文本编辑器，输入一两个句子，然后将其保存为 friends.dat。

如果您使用的是本书配套资源中的 friend.dat，应用程序 BufferConverter 的输出将如下：

```

Original byte data:
70 114 105 101 110 100 115 44 32 82 111 109 97 110 115 44 32
99 111 117 110 116 114 121 109 101 110 44 32 108 101 110 100
32 109 101 32 121 111 117 114 32 101 97 114 115 46 13 10 13
10

```

```

New character data:
Friends, Romans, countrymen, lend me your ears.

```

应用程序 BufferConverter 使用本章介绍的技术来读取数据，并将其显示为 byte 和字符。然而，使用旧的输入/输出包 java.io，也能够完成这项任务。

因此，您可能会问，为何要学习这个新的输入/输出包？

原因之一是，缓冲区让您能够以快得多的速度操纵大量的数据，下一节将介绍另一个原因。

## 17.2.4 网络通道

java.nio 包中最流行的特性可能是对通过网络连接的非阻断（non-blocking）输入/输出的支持。

在 Java 中，阻断（blocking）指的是程序中的语句必须执行完毕后，才执行其他操作。前面完成的所有套接字编程都只使用了阻断方法。例如，在应用程序 TimeServer 中，当服务器套接字的 accept() 方法被调用时，仅当客户建立连接后，程序才执行其他操作。

正如您认为的，网络程序等待特定的语句被执行是一种问题，因为可能导致错误的因素很多。连接可能断开，服务器可能离线。由于被阻断的语句需要等待某种事情发生，因此套接字连接的速度可能很慢。

例如，使用 HTTP 连接读取数据并将其放到缓冲区中的客户端应用程序，可能要等待缓冲区被填满，即使没有其他的数据需要发送。由于被阻断的语句不能执行完毕，因此程序好像已停止。

有了 java.nio 包，您可以创建网络连接，并使用非阻断方法通过网络连接进行读写。

其工作原理如下：

- 将套接字通道与输入/输出流关联起来，
- 对通道进行配置，使之能够识别您要监视的网络事件——建立新连接、通过通道读/写数据；
- 调用一个方法来打开通道；
- 由于该方法是非阻断的，因此程序将继续执行，让您能够处理其他任务；
- 如果监视的网络事件发生，则调用与该事件相关联的方法，以通知程序。

这类似 Swing 中的用户界面组件编程。界面组件同一个或多个事件监听器相关联，并被加入到容器中。界面组件收到监听器监视的输入后，将调用事件处理方法。在此之前，程序可以处理其他任务。

要使用非阻塞输入和输出，必须利用通道，而不是流。

#### 非阻塞套接字客户端和服务端

要开发非阻塞客户端或服务端软件，首先需要创建一个对象，该对象表示您要连接到的 Internet 地址。为此，可以使用 java.net 包中的 InetSocketAddress 类。

如果服务器是使用主机名标识的，则可以调用 InetSocketAddress(String, int)，其中的两个参数分别是服务器的名称和端口号。

如果服务器是使用 IP 地址标识的，则可使用 java.net 包中的 InetAddress 类来标识主机。为此，调用静态方法 InetAddress.getByName(String)，并将主机的 IP 地址作为参数传递给它。该方法返回一个表示地址的 InetAddress 对象，您可以将该对象作为参数来调用 InetSocketAddress(InetAddress, int) 方法，其中第二个参数是服务器的端口号。

要建立非阻塞连接，必须使用套接字通道——java.nio 包中的一个新类。要创建这种通道，可调用 SocketChannel 类的静态方法 open()。

套接字通道可被配置为进行阻塞通信或非阻塞通信。要设置为非阻塞通道，可调用通道的 configureBlocking(boolean) 方法，并将参数 false 传递给它；要设置为阻塞通道，则传递参数 true。

配置好通道后，可调用其 connect(InetSocketAddress) 方法来连接到套接字。

对于阻塞通道，connect() 方法将试图建立到服务器的连接，并等待这项操作的完成，然后返回 true，指出已成功地建立连接。

对于非阻塞通道，connect() 方法将立刻返回，并返回 false。要监视通道发生的事件，并对事件进行响应，必须使用通道监听对象 Selector。

Selector 对象跟踪套接字通道（或属于 SelectableChannel 的子类的通道）发生的事件。

要创建 Selector，可调用其 open() 方法，如下面的语句所示：

```
Selector monitor = Selector.open();
```

使用 Selector 时，必须指出要监视的事件。为此，可以调用通道的 register(Selector, int, Object) 方法。register() 方法的 3 个参数的含义如下：

- 您创建的用于监视通道的 Selector 对象；
- 表示要监视的事件的 int 值（也叫选定键值）；
- 随键值一起被传递的 Object，如果没有，则为 null。

对于第二个参数，使用 SelectionKey 类的类变量（而不是整数值）将更容易。这些类变量是：SelectionKey.OP\_CONNECT（监视连接操作）、SelectionKey.OP\_READ（监视读操作）、SelectionKey.OP\_WRITE（监视写操作）。

下面的语句创建了一个名为 spy 的 Selector，用于监视套接字通道的读操作：

```
Selector spy = Selector.open();
channel.register(spy, SelectionKey.OP_READ, null);
```

要监视多种事件，可将 SelectionKey 类的类变量加起来，例如：

```
Selector spy = Selector.open();
channel.register(spy, SelectionKey.OP_READ + SelectionKey.OP_WRITE, null);
```

设置好通道和选择器（selector）后，可以调用选择器的 select() 或 select(long) 方法，来等待事件的发生。

select() 是一个阻塞方法，它等待通道发生某种事件。

Select(long) 也是一个阻塞方法，它等待通道发生某种事件或过去指定的时间（单位为毫秒）。

这两个方法都返回发生的事件数，如果没有发生任何事件，则返回 0。可以在 while 循环中调用

select()方法,直到某种事件发生后才退出循环。

事件发生后,可以调用选择器的 selectedKeys()方法来了解事件的细节,该方法返回一个 Set 对象,其中包含每个事件的细节。

该 Set 对象的用法与其他集合 (set) 相同——创建一个 Iterator,然后使用其 hasNext() 和 next() 方法来遍历集合。

集合的 next() 方法返回一个对象,您应将其强制转换为 SelectionKey 对象。该对象表示通道发生的事件。

在客户程序中,可以使用 SelectionKey 类的 3 个方法来确定键值: isReadable()、isWritable() 和 isConnectible()。这些方法都返回一个布尔值 (将数据写入到服务器时,需要使用第 4 个方法: isAcceptable())。

取回集合中的键值后,调用键值的 remove() 方法,以指出您要对它执行某种操作。

关于事件,需要了解的最后一项内容是,它发生在哪个通道上。为此,可以调用键值的 channel() 方法,它返回相关联的 SocketChannel。

如果事件与连接相关,则使用通道之前,必须确保该连接已经建立。为此,可以调用键值的 isConnectionPending() 方法。如果连接仍在建立中,该方法返回 true; 如果已经建立,则返回 false。

要处理仍在建立中的连接,可调用套接字的 finishConnect() 方法,它试图完成连接。

使用非阻塞套接字通道时,涉及 java.nio 和 java.net 包中大量新类的交互。

为让读者更清晰地了解这些类是如何协同工作的,本章的最后一个项目 NewFingerServer (这是一个 Web 应用程序) 使用了非阻塞套接字通道来处理 finger 请求。

请输入程序清单 17.5 中的代码,将其保存为 NewFingerServer.java,然后编译它。

**程序清单 17.5 完整的 NewFingerServer.java 源代码**

```

1: import java.io.*;
2: import java.net.*;
3: import java.nio.*;
4: import java.nio.channels.*;
5: import java.util.*;
6:
7: public class NewFingerServer {
8:
9:     public NewFingerServer() {
10:         try {
11:             // Create a nonblocking server socket channel
12:             ServerSocketChannel sockChannel = ServerSocketChannel.open();
13:             sockChannel.configureBlocking(false);
14:
15:             // Set the host and port to monitor
16:             InetSocketAddress server = new InetSocketAddress(
17:                 "localhost", 79);
18:             ServerSocket socket = sockChannel.socket();
19:             socket.bind(server);
20:
21:             // Create the selector and register it on the channel
22:             Selector selector = Selector.open();
23:             sockChannel.register (selector, SelectionKey.OP_ACCEPT);
24:
25:             // Loop forever, looking for client connections
26:             while (true) {
27:                 // Wait for a connection
28:                 selector.select();
29:
30:                 // Get list of selection keys with pending events
31:                 Set keys = selector.selectedKeys();
32:                 Iterator it = keys.iterator();
33:
34:                 // Handle each key
35:                 while (it.hasNext()) {
36:

```

```

37:         // Get the key and remove it from the iteration
38:         SelectionKey selKey = (SelectionKey) it.next();
39:
40:         it.remove();
41:         if (selKey.isAcceptable()) {
42:
43:             // Create a socket connection with the client
44:             ServerSocketChannel selChannel =
45:                 (ServerSocketChannel) selKey.channel();
46:             ServerSocket selSocket = selChannel.socket();
47:             Socket connection = selSocket.accept();
48:
49:             // Handle the finger request
50:             handleRequest(connection);
51:             connection.close();
52:         }
53:     }
54: }
55: } catch (IOException ioe) {
56:     System.out.println(ioe.getMessage());
57: }
58: }
59:
60: private void handleRequest(Socket connection) throws IOException {
61:
62:     // Set up input and output
63:     InputStreamReader isr = new InputStreamReader (
64:         connection.getInputStream());
65:     BufferedReader is = new BufferedReader(isr);
66:     PrintWriter pw = new PrintWriter(new
67:         BufferedOutputStream (connection.getOutputStream()),
68:         false);
69:
70:     // Output server greeting
71:     pw.println("Nio Finger Server");
72:     pw.flush();
73:
74:     // Handle user input
75:     String outLine = null;
76:     String inLine = is.readLine();
77:
78:     if (inLine.length() > 0) {
79:         outLine = inLine;
80:     }
81:     readPlan(outLine, pw);
82:
83:     // Clean up
84:     pw.flush();
85:     pw.close();
86:     is.close();
87: }
88:
89: private void readPlan(String userName, PrintWriter pw) {
90:     try {
91:         FileReader file = new FileReader (userName + ".plan");
92:         BufferedReader buff = new BufferedReader(file);
93:         boolean eof = false;
94:
95:         pw.println("\nUser name: " + userName + "\n");
96:
97:         while (!eof) {
98:             String line = buff.readLine();
99:
100:             if (line == null)
101:                 eof = true;
102:             else
103:                 pw.println(line);
104:         }
105:
106:         buff.close();
107:     } catch (IOException e) {
108:         pw.println("User " + userName + " not found.");

```

```

109:     }
110: }
111:
112: public static void main(String[] arguments) {
113:     NewFingerServer nio = new NewFingerServer();
114: }
115: }

```

该 finger 服务器要求有一个或多个以纯文本方式存储的用户.plan 文件，它们的文件名为 username.plan，如 linus.plan、lucy.plan 和 franklin.plan。运行这个服务器程序之前，创建一个或多个.plan 文件，并将它们保存到 NewFingerServer.class 所在的文件夹中。

然后，运行该服务器，且不提供任何参数：

```
java NewFingerServer
```

该应用程序将等待 finger 请求，创建一个非阻塞服务器套接字通道，并注册一种选择器寻找的键值 (key)：连接事件。

在从 25 行开始的 while 循环中，服务器调用 Selector 对象的 select() 方法来确定选择器是否收到键值——这种情况将在 finger 客户试图建立连接时发生。如果收到了，select() 将返回键值数，而循环中的语句将被执行。

建立连接后，创建一个缓冲阅读器，用于对.plan 文件的请求。该命令的语法是被请求的.plan 文件的用户名。

## 17.3 总结

本章介绍了如何结合使用 URL、URL 连接和输入流来将万维网上的数据读取到程序中。

网络技术很有用。程序 WebReader 一个是基本的 Web 浏览器，它能够将网页或 RSS 文件加载到 Java 程序中，并将其显示出来，虽然它们没有对 HTML 标记进行解析，而只是显示 Web 服务器提供的原始文本。

您创建了一个套接字应用程序，它实现了 finger 协议的基本功能。finger 协议是一种获取 Internet 上用户信息的方式。

您还学习了如何使用 java.nio 包中的非阻塞技术来编写客户程序和服务器程序。

为使用非阻塞技术，您学习了新的 Java 网络包中的基本类：缓冲区、字符编码器和解码器、套接字通道以及选择器。

## 17.4 问与答

问：如何进行 POST 表单提交？

答：要用 POST 来发送表单，您必须模仿浏览器的行为。创建一个表示表单提交地址（如 http://www.example.com/cgi/mail2.cgi）的 URL 对象，然后调用该对象的 openConnection() 方法来创建一个 URLConnection 对象。调用连接对象的 setDoOutput() 方法来指出您要将数据发送到该 URL，然后将一系列用 & 符号分隔的名称-值对（其中存储了数据）发送给该连接。

例如，如果 mail2.cgi 表单是一个 CGI 程序，它发送包括 name、subject、email 和 comments 字段的邮件，且您创建了与该 CGI 程序相连的、名为 pw 的 PrintWriter 流，则可以使用下面的语句将信息递交给它：

```

pw.print("name=YourName&subject=Book&email=you@yourdomain.com&"
+ "comments= A+POST+example ");

```



## 17.5 小测验

请回答下述问题，以复习本章介绍的内容。

### 17.5.1 问题

- 下列哪项不是新的 java.nio 及其相关包的优点？
  - 通过使用缓冲区，能够快速处理大量的数据；
  - 应用程序的网络连接可以是非阻塞的，因此更可靠；
  - 通过网络读写数据时不再需要流。
- 在 finger 协议中，哪个程序请求有关用户的信息？
  - 客户程序；
  - 服务器程序；
  - 都可以。
- 要将网页中的数据加载到 Java 应用程序中，哪种方式最佳？
  - 创建一个 Socket，并使用该套接字来创建一个输入流；
  - 创建一个 URL，并使用该 URL 来创建一个 URLConnection；
  - 使用方法 toString() 来加载页面。

答案

- c, java.nio 包和流协同工作，它不能替代流。
- a, 客户请求信息，服务器发回信息，对请求进行响应。这是传统的客户/服务器应用程序的工作原理，虽然有些程序可同时充当客户和服务器。
- b, 套接字适用于低级连接，如实现新的协议时。对于诸如 HTTP 等现有的协议，有更适合的类——URL 和 URLConnection。

### 17.5.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行检测。

```
import java.nio.*;

public class ReadTemps {
    public ReadTemps() {
        int[] temperatures = { 78, 80, 75, 70, 79, 85, 92, 99, 90, 85, 87 };
        IntBuffer tempBuffer = IntBuffer.wrap(temperatures);
        int[] moreTemperatures = { 65, 44, 71 };
        tempBuffer.put(moreTemperatures);
        System.out.println("First int: " + tempBuffer.get());
    }
}
```

上述应用程序运行时，其输出为：

- First int:78;
- First int:71;
- First int:70;
- 都不是。

答案 d

## 17.6 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 编写一个应用程序，将您喜欢的网页存储到计算机中，以便离线时能够阅读它们。
2. 编写一个程序，它接受 finger 请求、查找与请求的用户名匹配的.plan 文件。如果找到，则将其发回；否则，发送消息“user not found”。



## 第 18 章

# 使用 JDBC 访问数据库

几乎所有的 Java 程序都需要以某种方式来处理数据。到现在为止，您使用了基本数据类型、对象、数组、链表和其他数据结构来表示数据。

本章将以更复杂的方式来处理数据。Java Database Connectivity (JDBC) 是一个类库，它将 Java 程序连接到关系型数据库。

Java 6 包含 Java BD，这是一个小型关系型数据库，它是 Java 开发包的一部分，使得在应用程序中集成数据库比以前任何时候都容易。

本章介绍 JDBC 和可扩展标记语言 (XML)，包括如下主题：

- 通过 JDBC 驱动程序使用各种关系型数据库；
- 使用结构化查询语言 (SQL) 访问数据库；
- 使用 SQL 和 JDBC 读取数据库中的记录；
- 使用 SQL 和 JDBC 将记录加入到数据库中；
- 创建一个 Java DB 数据库并读取其中的记录。

### 18.1 JDBC

JDBC (Java Database Connectivity, Java 数据库连接) 是一组类，可用于开发这样的客户/服务器应用程序，即使用 Microsoft、Sybase、Oracle、Informix 开发的数据库以及其他数据源。

通过 JDBC，可以在 Java 程序中使用相同的类和方法来读写记录以及执行其他数据库访问操作。名为驱动程序的类是连接到数据源的桥梁——对于每种流行的数据库，都有相应的驱动程序。

客户/服务器软件将使用信息的用户与信息提供方连接起来，是最常见的编程方式之一。您在网上冲浪时都使用了它：Web 浏览器使用统一资源定位符 (Uniform Resource Locator, URL) 请求网页、图像文件和其他文档，如果找到了，服务器程序将请求的信息提供给客户端。

数据库程序员面临的最大挑战是：数据库格式众多，且每种格式都使用专用的方法来访问数据。

为简化关系型数据库程序的用法，人们开发了一种名为 SQL 的标准语言，从而避免学习针对每种数据库格式的数据库查询语言。Java 6 包含的数据库 Java DB 也支持 SQL。

在数据库编程中，请求数据库中的记录被称为查询。使用 SQL，您可以将复杂的查询发送给数据库，并按指定的顺序获得查找的记录。

来看一个例子，某数据库程序员在一个学生贷款公司工作，他需要准备一份有关拖欠贷款长的贷款人的报告。该程序员可以使用 SQL 在数据库中查询最近一次偿还在 180 天以前，且拖欠金额超过 \$0.00 的记录。SQL 还可用于控制记录被返回的顺序，因此程序员可以某种顺序（如社会保险号、借方姓名、所欠金额或贷款数据库中的其他字段）排列记录。

所有这些操作都可以通过 SQL 来完成，程序员无需使用与数据库格式相关联的专用语言。

**警告**

很多数据库格式都支持 SQL，因此从理论上说，对于每种支持 SQL 的数据库工具，您都能使用相同的 SQL 命令。然而，当您通过 SQL 访问特定数据库格式时，还是需要了解它的一些特征。

SQL 是访问关系型数据库的行业标准。JDBC 支持 SQL，让开发人员能够使用各种数据库格式，而无需知道底层数据库的具体细节。它还允许使用针对特定数据库格式的数据库查询。

JDBC 类库通过 SQL 访问数据库的方式与现有的数据库开发技术类似，因此使用 JDBC 与 SQL 数据库交互与使用传统的数据工具没有太大的不同。使用过数据库的 Java 程序员在使用 JDBC 时将轻车熟路。JDBC API 已被行业领头羊们广泛认可，这包括一些开发工具厂商，他们宣传将在其开发产品支持 JDBC。

对于每种常见的数据库使用任务，JDBC 库中都有相应的类：

- 连接到数据库；
- 使用 SQL 创建语句；
- 在数据库中执行 SQL 查询；
- 查看结果。

这些 JDBC 类都位于 java.sql 包中。

## 数据库驱动程序

使用 JDBC 类的 Java 程序可以遵循这样的编程模型：执行 SQL 语句、处理查询结果。数据库的格式及其针对的平台是无关紧要的。

这种平台和数据库无关性是通过驱动程序管理器实现的。JDBC 类库中的类主要依赖于驱动程序管理器，后者跟踪访问数据库记录所需的驱动程序。对于每种数据库格式，都需要不同的驱动程序。有时候，对于同一种格式的不同版本，需要使用多个不同的驱动程序。Java DB 也有驱动程序。

JDBC 还包括一个这样的驱动程序，即能够将 JDBC 和另一种名为 ODBC 的数据库连接标准连接起来。

## 18.2 JDBC-ODBC 桥

ODBC 是 Microsoft 用于访问 SQL 数据库的通用接口，在 Windows 系统上，它是由 ODBC 数据源管理器管理的。

在 Windows 系统中，可从控制面板运行它。为此，在大多数 Windows 版本中，可单击“开始”按钮并选择“设置”>“控制面板”，然后，双击“ODBC 数据源”。在 Windows XP 中，可单击“开始”按钮并选择“控制面板”。如果当前处于分类视图，则单击“性能和维护”，然后依次双击“管理工具”和“数据源 (ODBC)”；如果处于经典视图，则直接依次双击“管理工具”和“数据源 (ODBC)”。

管理器添加 ODBC 驱动程序、配置驱动程序使之能够处理特定的数据库文件并记录 SQL 的使用情况。图 18.1 是 Windows 系统中的 ODBC 数据源管理器。

在图 18.1 中，选项卡 Drivers 中列出了系统中所有的 ODBC 驱动程序。其中的很多针对某数据库公司的格式，如 Microsoft Access Driver。

JDBC-ODBC 桥使得 JDBC 驱动程序可用作 ODBC 驱动程序，这是通过将 JDBC 方法调用转换为 ODBC 函数调用实现的。

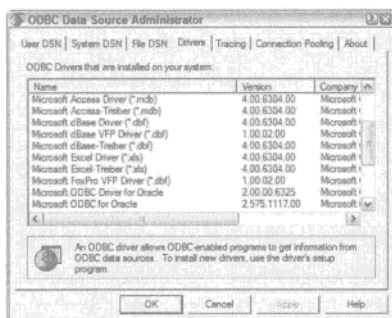


图 18.1 Windows 系统中的 ODBC 数据源管理器

要使用 JDBC-ODBC 桥，需要 3 样东西。

- Java 中的 JDBC-ODBC 桥驱动程序：sun.jdbc.odbc.JdbcOdbcDriver。
- 一个 ODBC 驱动程序。
- 一个 ODBC 数据源，该数据源已通过诸如 ODBC 数据源管理等软件关联到驱动程序。

ODBC 数据源可从数据库程序中建立。例如，在 Lotus Approach 中创建新的数据库文件时，用户可以将其与 ODBC 驱动程序关联起来。

对于所有的 ODBC 数据源，都必须给它指定一个简短的描述性名称。在 Java 程序中连接到数据源指向的数据库时，需要使用这个名称。

在 Windows 系统中，选择 ODBC 驱动程序并创建数据库后，它们将显示在 ODBC 数据源管理器中。图 18.2 显示了一个这样的例子，其中的数据源名为 WorldEnergy。

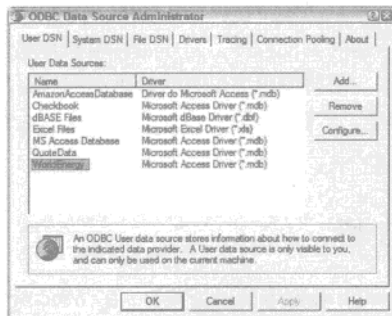


图 18.2 ODBC 数据源管理器中的数据源列表

从图 18.2 可知，数据源 WorldEnergy 被关联到 Microsoft Access Driver。

#### 注意

大多数 Windows 数据库程序都包括一个或多个与其格式相对应的 ODBC 驱动程序。Microsoft Access 中包含可用于连接到 Access 数据库文件的 ODBC 驱动程序。

### 18.2.1 连接到 ODBC 数据源

本章要创建的第一个项目是一个这样的 Java 应用程序，即它使用 JDBC-ODBC 桥连接到 Microsoft Access 文件。

这个程序使用的 Access 文件是 world20.mdb，这是一个世界能源统计数据数据库，由美国能源信息管理局公布。该数据库中的 Coal 表包括以下字段：

- Country (国家及地区);
- Year;
- Anthracite Production。

要使用这个数据库,您的系统上必须有支持 Microsoft Access 文件的 ODBC 驱动程序。使用 ODBC 数据源管理器 (或类似的程序,如果您使用的不是 Windows 系统) 创建一个新的 ODBC 数据源,并将其关联到 world20.mdb。

可能还需要进行其他设置工作,这取决于您的系统上的 ODBC 驱动程序。有关这方面的信息,请参考 ODBC 驱动程序中的文档。

将 world20.mdb 下载到计算机中或在系统上找到其他与 ODBC 驱动程序兼容的数据库后,要通过 JDBC-ODBC 访问该文件,需要做的最后一项准备工作是创建一个数据源,并将其与该文件关联起来。与 Java 中的其他输入/输出类不同, JDBC 并不使用文件名来标识数据文件并使用其内容,而是使用诸如 ODBC 数据源管理等工具来给 ODBC 资源命名,并指定其所在的文件夹。

在 ODBC 数据源管理器中,单击“用户 DSN”标签,以查看当前可用的数据源列表。要新增一个数据源并将其与 world20.mdb (或您自己的数据库) 关联起来,可单击按钮“添加”,然后选择一个 ODBC 驱动程序并单击“完成”按钮。

这将打开一个“安装”对话框。通过该对话框,您可以提供名称、简短的描述以及关于该数据库的其他信息。单击“选取”按钮,以查找并选择数据库文件。

图 18.3 显示了在 ODBC 数据源管理器中将 world20.mdb 设置为一个数据源的情况。

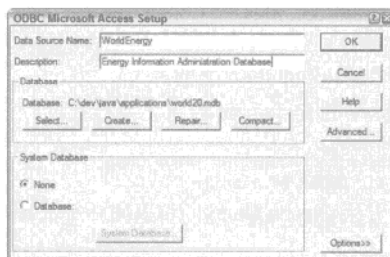


图 18.3 驱动程序“安装”窗口

将数据库与 ODBC 数据源关联起来后,如果您熟悉 SQL,则在 Java 程序中使用它将是比较容易的。

在 JDBC 程序中,首先需要装载将用于连接到数据源的驱动程序。驱动程序是使用方法 `Class.forName(String)` 来装载的。Class 位于 `java.lang` 包中,可用于将类装载到 Java 解释器中。方法 `forName(String)` 装载字符串参数指定的类,并可能引发 `ClassNotFoundException` 异常。

所有使用 ODBC 数据源的程序都要用到 `sun.jdbc.odbc.JdbcOdbcDriver`——Java 中的一个 JDBC-ODBC 桥驱动程序。要将这个类装载到 Java 解释器中,可使用如下语句:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

装载驱动程序后,可以使用 `java.sql` 包中的 `DriverManager` 类来建立到数据源的连接。

`DriverManager` 的方法 `getConnection(String, String, String)` 可用于建立这种连接。它返回 `Connection` 对象引用,该对象表示一个活动的数据库连接。

这个方法的 3 个参数如下:

- 名称,指定数据源和用于连接到该数据源的数据库连接类型;
- 用户名;
- 密码;

仅当数据源通过用户名和密码进行保护时，才需要后面两个参数；否则，这两个参数可设置为空字符串（""）。

使用 JDBC-ODBC 桥时，应在数据源名称前加上 jdbc:odbc:，这指出了使用的数据库连接类型。

下面的语句可用于连接名为 Payroll 的数据源，其中用户名为 Doc，密码为 lrover1：

```
Connection payday = DriverManager.getConnection(
    "jdbc:odbc:Payroll", "Doc", "lrover1");
```

建立连接后，每当需要检索该连接的数据源或将信息存储到其中时，都可以重用它。

如果使用数据源时发生了错误，方法 `getConnection` 以及数据源的其他所有方法都可能引发 `SQLException` 异常。SQL 有自己的错误消息，它们将作为 `SQLException` 对象的一部分而被传递。

### 1. 使用 SQL 从数据库检索数据

在 Java 中，SQL 语句是用 `Statement` 对象表示的。`Statement` 是一个接口，因此不能被实例化。然而 `Connection` 对象的 `createStatement()` 方法返回一个实现了这种接口的对象，正如下例所示：

```
Statement lookSee = payday.createStatement();
```

有了 `Statement` 对象后，可以调用其 `executeQuery(String)` 方法来执行 SQL 查询。其中的 `String` 参数是一个符合 SQL 语法的 SQL 查询。

#### 警告

本章不打算介绍 SQL，这是一种内容丰富的数据检索和存储语言。

下面的 SQL 查询可用于数据库 `world20.mdb` 中的 `Coal` 表：

```
SELECT Country, Year, 'Anthracite Production' FROM Coal
WHERE (Country Is Not Null) ORDER BY Year
```

该 SQL 查询检索数据库中 `Country` 字段不为空的各个记录的多个字段。被返回的记录将按 `Country` 字段排序，因此 `Afghanistan` 将排在 `Burkina Faso` 之前。

下面的 Java 语句在名为 `Looksee` 的 `Statement` 对象上执行查询：

```
ResultSet set = looksee.executeQuery(
    "SELECT Country, Year, 'Anthracite Production' FROM Coal "
    + "WHERE (Country Is Not Null) ORDER BY Year");
```

如果 SQL 查询的语法正确，方法 `executeQuery()` 将返回一个 `ResultSet` 对象，其中包含从数据源中取回的所有记录。

#### 注意

要将记录添加到数据库中而不是检索数据库中的记录，应调用 `Statement` 对象的 `executeUpdate()` 方法，这将在后面介绍。

`executeQuery()` 返回的 `ResultSet` 指向取回的第一条记录。可调用 `ResultSet` 的下述方法来获取当前记录中的信息。

- `getDate(String)`：返回指定字段中的 `Data` 值（使用 `java.sql` 而不是 `java.util` 包中的 `Data` 类）。
- `getDouble(String)`：返回指定字段中的 `double` 值。
- `getFloat(String)`：返回指定字段中的 `float` 值。
- `getInt(String)`：返回指定字段中的 `int` 值。
- `getLong(String)`：返回指定字段中的 `long` 值。
- `getString(String)`：返回指定字段中的 `String`。

这些只是接口 `ResultSet` 中最简单的方法。您应使用的方法取决于数据库创建时数据字段的格式，虽然诸如 `getString()` 和 `getInt()` 等方法在检索记录中的信息方面可能更灵活。

您也可以将整数（而不是字符串）作为参数传递给上述方法，例如 `getString(5)`。该整数指定要检索哪个字段（1 表示第一个字段，2 表示第二个字段，以此类推）。

当您试图从结果集中检索信息时，如果发生错误，将引发 `SQLException`。要获取有关错误的更详

细的信息，可调用该异常的 `getSQLState()` 和 `getErrorCode()` 方法。

从记录中获取所需的信息后，可以调用 `ResultSet` 对象的 `next()` 方法来移到下一条记录。移动时，如果超出结果集末尾，该方法将返回布尔值 `false`。

通常，可以从头到尾遍历整个结果集一次，之后便不能再检索其内容。

使用完到数据源的连接后，可以调用 `close()` 方法（不提供任何参数）来关闭它。

程序清单 18.1 的应用程序 `CoalReporter` 使用 JDBC-ODBC 桥和 SQL 语句来检索能源数据库中的一些记录。对于 SQL 语句指定的每条记录，都将检索其中的 4 个字段：FIPS、Country、Year 和 Anthracite Production。结果集按 Year 字段排序，而字段被显示在标准输出中。

程序清单 18.1 完整的 `CoalReporter.java` 源代码

```
1: import java.sql.*;
2:
3: public class CoalReporter {
4:     public static void main(String[] arguments) {
5:         String data = "jdbc:odbc:WorldEnergy";
6:         try {
7:             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
8:             Connection conn = DriverManager.getConnection(
9:                 data, "", "");
10:            Statement st = conn.createStatement();
11:            ResultSet rec = st.executeQuery(
12:                "SELECT * " +
13:                "FROM Coal " +
14:                "WHERE " +
15:                "(Country=' " + arguments[0] + "') " +
16:                "ORDER BY Year");
17:            System.out.println("FIPS\tCOUNTRY\tYEAR\t" +
18:                "ANTHRACITE PRODUCTION");
19:            while(rec.next()) {
20:                System.out.println(rec.getString(1) + "\t" +
21:                    rec.getString(2) + "\t\t" +
22:                    rec.getString(3) + "\t" +
23:                    rec.getString(4));
24:            }
25:            st.close();
26:        } catch (SQLException s) {
27:            System.out.println("SQL Error: " + s.toString() + " " +
28:                s.getErrorCode() + " " + s.getSQLState());
29:        } catch (Exception e) {
30:            System.out.println("Error: " + e.toString() +
31:                e.getMessage());
32:        }
33:    }
34: }
```

运行该程序时，必须指定一个参数，它指定应检索数据库中 Country 字段为相应值的记录，如下述 JDK 示例所示：

```
java CoalReporter Poland
```

如果运行该应用程序时，指定的参数为 Poland，则输出如下：

FIPS	COUNTRY	YEAR	ANTHRACITE PRODUCTION
PL	Poland	1990	0.0
PL	Poland	1991	0.0
PL	Poland	1992	0.0
PL	Poland	1993	174.165194805424
PL	Poland	1994	242.50849909616
PL	Poland	1995	304.237935229728
PL	Poland	1996	308.64718066784
PL	Poland	1997	319.67029426312
PL	Poland	1998	319.67029426312

请尝试使用其他国名（如 France、Swaziland、New Zealand 等）为参数来运行该程序。如果国名中包含空格，别忘了用引号将国名括起来。



## 2. 使用 SQL 将数据写入到数据库中

在应用程序 CoalReporter 中, 您使用 SQL 语句来检索数据库中的数据。SQL 语句被加工为一个字符串, 如下所示:

```
SELECT * FROM Coal WHERE (Country='Swaziland') ORDER BY YEAR
```

这是一种常见的使用 SQL 的方式。您也可以编写这样的程序, 即让用户输入一个 SQL 查询, 然后显示查询结果 (虽然这可能不是一个好主意——SQL 查询可用于删除记录、表, 甚至整个数据库)。

java.sql 包还支持另一种创建 SQL 语句的方式: 准备好的语句。

准备好的语句 (prepared statement) 是用 PreparedStatement 类表示的, 它是执行前将被编译的 SQL 语句。这加快了语句返回数据的速度, 因此对于程序中重复执行的 SQL 语句, 这是一种更好的选择。

### 提示

在 Windows 系统中, 准备好的语句还有另一个优点: 使得可以使用 JDBC-ODBC 桥驱动程序将数据写入到 Microsoft Access 数据库中。多年来, 我一直无法在 Java 中使用语句将数据写入到 Access 数据库中, 但使用准备好的语句时, 则没有任何问题。

要创建准备好的语句, 可调用连接的 PreparedStatement(String) 方法, 并将一个字符串传递给它。该字符串指出了 SQL 语句的结构。

要指出结构, 可编写一条 SQL 语句, 并将其中的参数替换为问号。

下面的例子演示了如何调用连接对象的 PreparedStatement 方法:

```
PreparedStatement ps = cc.prepareStatement(
    "SELECT * FROM Coal WHERE (Country='?') ORDER BY YEAR");
```

下面的示例使用了多个问号:

```
PreparedStatement ps = cc.prepareStatement(
    "INSERT INTO BOOKDATA VALUES(?, ?, ?, ?, ?, ?, ?)");
```

这些 SQL 语句中的问号为数据占位符。执行这些语句之前, 您必须使用 PreparedStatement 类的方法将数据加入到这些地方。

要将数据加入到准备好的语句中, 必须调用一个方法, 并将占位符的位置和要插入的数据作为参数传递给它。

例如, 要将字符串 "Swaziland" 插入到第一条准备好的语句中, 可以调用方法 setString(int, String):

```
ps.setString(1, "Swaziland");
```

第一个参数指定了占位符的位置 (从左到右进行编号): 1 表示第一个问号, 2 表示第二个问号, 依此类推。

第二个参数是要插入到指定位置的数据。

可用的方法有:

- setAsciiStream(int, InputStream, int): 将指定的 InputStream (表示一个 ASCII 字符流) 插入到第一个参数指定的位置; 第三个参数指定插入流中的多少个字节。
- setBinaryStream(int, InputStream, int): 将指定的 InputStream (表示一个字节流) 插入到第一个参数指定的位置; 第三个参数指定插入流中的多少个字节。
- setCharacterStream(int, Reader, int): 将指定的 Reader (表示一个字符流) 插入到第一个参数指定的位置; 第三个参数指定插入流中的多少个字符。
- setBoolean(int, boolean): 将一个布尔值插入到 int 参数指定的位置。
- setByte(int, byte): 将一个 byte 值插入到指定的位置。
- setBytes(int, byte[]): 将一个 byte 数组插入到指定的位置。
- setDate(int, Date): 将一个 Date 对象 (位于 java.sql 包中) 插入到指定的位置。
- setDouble(int, double): 将一个 double 值插入到指定的位置。
- setFloat(int, float): 将一个 float 值插入到指定的位置。

- `setInt(int, int)`: 将一个 `int` 值插入到指定的位置。
- `setLong(int, long)`: 将一个 `long` 值插入到指定的位置。
- `setShort(int, short)`: 将一个 `short` 数组插入到指定的位置。
- `setString(int, String)`: 将一个 `String` 值插入到指定的位置。

还有一个 `setNull(int, int)` 方法, 它将一个控制插入到第一个参数指定的位置。

`setNull()` 方法的第二个参数应为 `Types` 类 (位于 `java.sql` 包中) 的一个类变量, 它指出了该位置的 SQL 值的类型。

对于每种 SQL 数据类型, 都有一个相应的类变量。其中最常用的变量有: `BIGINT`、`BIT`、`CHAR`、`DATE`、`DECIMAL`、`DOUBLE`、`FLOAT`、`INTEGER`、`SMALLINT`、`TINYINT` 和 `VARCHAR`。

下面的代码将一个空的 `CHAR` 值加入到准备好的语句 `ps` 的第 5 个位置:

```
ps.setNull(5, Types.CHAR);
```

接下来的程序演示了如何使用准备好的语句将股价数据加入到数据库中。股价数据是从 Yahoo! (美国) 网站收集的。

为给炒股者提供服务, Yahoo! 网站为其股票报价页面中的每个股票简称提供了一个 `Download Spreadsheet` 链接。

要查看这种链接, 请进入 Yahoo! (美国) 的股票报价页面或直接进入下面这样的页面:

```
http://quote.yahoo.com/q?s=sunw&d=v1
```

在股价和成交量表格的下面, 有一个 `Download Spreadsheet` 链接。

对于 Sun Microsystems, 这种链接的内容如下:

```
http://download.finance.yahoo.com/d/quotes.csv?s=SUNW&f=s1d1t1c1ohgv&e=.csv
```

您可以单击该链接来打开文件或将其下载到您的计算机中。该文件只有一行, 其中包含最后一个交易日收盘时的股价和成交量。例如, 在 2007 年 2 月 23 日, Sun 的数据如下:

```
"SUNW",6.27,"2/23/2007","4:00pm",0.00,6.30,6.31,6.22,50254356
```

上述字段依次为股票简称、收盘价、日期、时间、涨跌幅、最低价、最高价、开盘价和成交量。

应用程序 `QuoteData` 使用这些字段 (时间字段除外, 该字段的用途不大, 因为其值总是为收盘时间)。

该程序执行了以下操作:

- 从命令行参数中获取股票简称;
- 创建一个 `QuoteData` 对象, 并将其实例变量 `ticker` 设置为股票简称;
- 调用该对象的 `retrieveQuote()` 方法从 Yahoo! (美国) 网站下载股票数据, 并将其作为一个 `String` 返回;
- 调用该对象的 `storeQuote()` 方法, 并将前述 `String` 作为参数传递给它, 该方法使用 `JDBC-ODBC` 连接将股票数据存储到数据库中。

要完成最后一项任务, 需要建立一个可通过 `JDBC-ODBC` 连接到的股票数据库, 用于收集这些数据。

Windows 用户可以使用本书配套资源 `quotedata.mdb`, 这是一个 Microsoft Access 2000 数据库, 可用于存储从 Yahoo! (美国) 网站下载的股票数据。下载该数据库 (或创建自己的数据库) 后, 使用 `ODBC` 数据源管理器创建一个新的数据源, 并将其与该数据库关联起来。该应用程序假设数据源的名称为 `QuoteData`。

请在编辑器中输入程序清单 18.2 中的代码, 并将其保存为文件 `QuoteData.java`。

#### 程序清单 18.2 完整的 `QuoteData.java` 源代码

```
1: import java.io.*;
2: import java.net.*;
3: import java.sql.*;
4: import java.util.*;
```

```

5:
6: public class QuoteData {
7:     private String ticker;
8:
9:     public QuoteData(String inTicker) {
10:         ticker = inTicker;
11:     }
12:
13:     private String retrieveQuote() {
14:         StringBuffer buf = new StringBuffer();
15:         try {
16:             URL page = new URL("http://quote.yahoo.com/d/quotes.csv?s=" +
17:                 ticker + "&f=s1d1t1c1ohgv&e=.csv");
18:             String line;
19:             URLConnection conn = page.openConnection();
20:             conn.connect();
21:             InputStreamReader in= new InputStreamReader(
22:                 conn.getInputStream());
23:             BufferedReader data = new BufferedReader(in);
24:             while ((line = data.readLine()) != null) {
25:                 buf.append(line + "\n");
26:             }
27:         } catch (MalformedURLException mue) {
28:             System.out.println("Bad URL: " + mue.getMessage());
29:         } catch (IOException ioe) {
30:             System.out.println("IO Error:" + ioe.getMessage());
31:         }
32:         return buf.toString();
33:     }
34:
35:     private void storeQuote(String data) {
36:         StringTokenizer tokens = new StringTokenizer(data, ",");
37:         String[] fields = new String[9];
38:         for (int i = 0; i < fields.length; i++) {
39:             fields[i] = stripQuotes(tokens.nextToken());
40:         }
41:         String datasource = "jdbc:odbc:QuoteData";
42:         try {
43:             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
44:             Connection conn = DriverManager.getConnection(
45:                 datasource, "", "");
46:             PreparedStatement prep2 = conn.prepareStatement(
47:                 "INSERT INTO " +
48:                 "Stocks(ticker, price, quoteDate, change, open, " +
49:                 "high, low, volume) " +
50:                 "VALUES(?, ?, ?, ?, ?, ?, ?, ?)");
51:             prep2.setString(1, fields[0]);
52:             prep2.setString(2, fields[1]);
53:             prep2.setString(3, fields[2]);
54:             prep2.setString(4, fields[4]);
55:             prep2.setString(5, fields[5]);
56:             prep2.setString(6, fields[6]);
57:             prep2.setString(7, fields[7]);
58:             prep2.setString(8, fields[8]);
59:             prep2.executeUpdate();
60:             conn.close();
61:         } catch (SQLException sqe) {
62:             System.out.println("SQL Error: " + sqe.getMessage());
63:         } catch (ClassNotFoundException cnfe) {
64:             System.out.println(cnfe.getMessage());
65:         }
66:     }
67:
68:     private String stripQuotes(String input) {
69:         StringBuffer output = new StringBuffer();
70:         for (int i = 0; i < input.length(); i++) {
71:             if (input.charAt(i) != '"') {
72:                 output.append(input.charAt(i));
73:             }
74:         }
75:         return output.toString();
76:     }

```



```

77:
78:     public static void main(String[] arguments) {
79:         if (arguments.length < 1) {
80:             System.out.println("Usage: java QuoteData tickerSymbol");
81:             System.exit(0);
82:         }
83:         QuoteData qd = new QuoteData(arguments[0]);
84:         String data = qd.retrieveQuote();
85:         qd.storeQuote(data);
86:     }
87: }

```

编译该程序后，连接到 Internet，并运行它。别忘了通过命令行参数指定一个有效的股票简称。要下载股票简称为 SUNW（Sun Microsystems）的股票数据，请执行下述命令：

```
java QuoteData SUNW
```

retrieveQuote() 方法（第 13~33 行）从 Yahoo!（美国）网站下载股票数据，并将其存储为一个字符串。该方法使用的技术在第 17 章中介绍过。

storeQuote() 方法（第 35~66 行）使用了本节介绍的 SQL 技术。

该方法首先以“,”为分隔符将股票数据分为一系列字符串标记（token），然后将这些标记存储到一个包含 9 个元素的 String 数组中。

这些字段在数组中存储顺序与 Yahoo! 网站上相同，依次为：股票简称、收盘价、日期、时间、涨跌幅、最低价、最高价、开盘价和成交量。

接下来，第 41~45 行使用 JDBC-ODBC 驱动程序建立了到数据源 QuoteData 的连接。

第 46~50 行使用该连接创建了一条准备好的语句。该语句使用 SQL 语句 INSERT INTO，这将导致数据被存储到数据库中。这里的数据库为 quotedata.mdb，而 INSERT INTO 语句指向的是该数据库中的 Stocks 表。

准备好的语句中有 8 个占位符。这里只需要 8 个而不是 9 个是因为该应用程序没有使用“时间”字段。

第 51~58 行调用一系列的 setString() 方法，将 String 数组中的元素插入到准备好的语句中。插入顺序与这些字段在数据库中的顺序相同，依次为股票简称、收盘价、日期、涨跌幅、最低价、最高价、开盘价和成交量。

在 Yahoo!（美国）网站中，有些字段为日期、浮点数和整数，因此您可能认为，对于这些数据，使用方法 setDate()、setFloat() 和 setInt() 可能更合适。

然而，当您使用 SQL 来处理数据库时，有些 Access 版本（包括 Access 2000）不支持其中的一些方法，虽然 Java 中有这些方法。如果您试图使用不被支持的方法，如 setFloat()，将发生 SQLException 异常。

发送 Access 字符串，让数据库程序自动将其转换为正确的格式将更容易。当您处理其他数据库时，情况也可能如此，对 SQL 的支持程度随数据库产品和 ODBC 而异。

编写好准备好的语句，并替换了其中的所有占位符后，第 59 行调用该语句对象的 executeUpdate() 方法。这可能将股票数据加入到数据库中，也可能引发 SQL 异常。私有方法 stripQuotes() 用于删除 Yahoo 网站的股票数据中的引号。第 39 行调用该方法来删除 3 个字段中的引号：股票简称、日期和时间。

### 3. 遍历结果集

默认情况下，结果集只允许被遍历一次：使用其方法 next() 来检索每条记录。通过修改创建语句和准备好的语句的方式，可以生成支持下述方法的结果集。

- afterLast(): 移到最后一条记录的后面。
- beforeFirst(): 移到第一条记录的前面。

- first(): 移到第一条记录。
- last(): 移到最后一条记录。
- previous(): 移到前一条记录。

调用数据库连接的方法 createStatement() 和 prepareStatement() 时, 如果通过参数指定了结果集的策略, 则生成的结果集将支持上述方法。

通常, 方法 createStatement() 不接受任何参数, 如下例所示:

```
Connection payday = DriverManager.getConnection(
    "jdbc:odbc:Payroll", "Doc", "Irover1");
Statement lookSee = payday.createStatement();
```

为使得到的结果集更为灵活, 可在调用方法 createStatement() 时指定 3 个整型参数, 这些参数指定了如何使用结果集。下面是上述语句的修订版本:

```
Statement lookSee = payday.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY,
    ResultSet.CLOSE_CURSORS_AT_COMMIT);
```

调用方法 prepareStatement (String, int, int, int) 时, 也可以指定上述参数, 这些参数位于语句字符串的后面。

ResultSet 类还包括其他类变量, 它们提供了有关结果集可被如何读取和修改的选项。

## 18.2.2 JDBC 驱动程序

创建使用 JDBC 驱动程序的 Java 程序与创建使用 JDBC-ODBC 桥的程序类似。

Java 6 包含关系型数据库 Java DB, 该数据库是基于开源数据库 Apache Derby 开发的, 并自带了驱动程序。对于更复杂的数据库, 有很多公司销售驱动程序或将其打包到商业产品中, 这包括 Informix、Oracle、Symantec、IBM 和 Sybase。当前可用的 JDBC 驱动程序列表可在 Sun 的网站找到, 其网址为 <http://java.sun.com/products/jdbc/drivers>。

### 注意

MySQL 数据库的开发人员提供了 Connector/J——Mark Matthews 开发的一个开源 JDBC 驱动程序。其中的一些驱动程序可下载用于评估。

要下载该驱动程序或获得有关它的更详细信息, 请访问 <http://www.mysql.com/downloads/connector/j/5.0.html>。

Java DB 位于 JDK 安装目录下的子文件夹 db 中。要开发连接到该数据库的应用程序, 必须使其驱动程序可用。为此, 一种方法是编辑环境变量 Classpath。

该数据库的驱动程序库位于子文件夹 db/lib 中的文件 derby.jar 中。如果 JDK 的安装目录为 C:\Program Files\jdk1.6.0, 则该库位于 C:\Program Files\jdk1.6.0\db\lib\derby.jar 中。请将该文件的完整引用 (包括文件夹和文件名) 添加到 Classpath 中。

建立 JDBC 数据源的步骤与 JDBC-ODBC 桥相同:

- 创建数据库;
- 将该数据库与 JDBC 驱动程序关联起来;
- 建立一个数据源, 这可能需要选择数据库格式、数据库服务器、用户名和密码。

程序清单 18.3 中的 Java 应用程序执行两项任务:

- 创建一个名为 Presidents 的 Java DB 数据库, 该数据库包含一个名为 contacts 的表, 其中包含 4 条记录;
- 读取该数据库表中的记录。

程序清单 18.3 完整的 Presidents.java 源代码

```

1: import java.io.*;
2: import java.sql.*;
3:
4: public class Presidents {
5:     String home, system;
6:
7:     public Presidents() {
8:         // set the database's directory
9:         home = System.getProperty("user.home", ".");
10:        system = home + File.separatorChar + ".database";
11:        System.setProperty("derby.system.home", system);
12:    }
13:
14:    public void createDatabase() {
15:        // create the database
16:        String data = "jdbc:derby:presidents;create=true";
17:        try {
18:            // load the driver
19:            Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
20:            // create the connection
21:            Connection conn = DriverManager.getConnection(data);
22:            Statement st = conn.createStatement();
23:            // create the contacts table
24:            int result = st.executeUpdate(
25:                "CREATE TABLE contacts ("
26:                + "dex INTEGER NOT NULL PRIMARY KEY "
27:                + "GENERATED ALWAYS AS identity "
28:                + "(START WITH 1, INCREMENT BY 1), "
29:                + "name VARCHAR(40), "
30:                + "address1 VARCHAR(40), "
31:                + "address2 VARCHAR(40), "
32:                + "phone VARCHAR(20), "
33:                + "email VARCHAR(40))");
34:            // insert four records into the new table
35:            result = st.executeUpdate(
36:                "INSERT INTO contacts (name, address1, address2, "
37:                + "phone, email) VALUES("
38:                + "'Jimmy Carter', "
39:                + "'Carter Presidential Center', "
40:                + "'1 Copenhill, Atlanta, GA 30307', "
41:                + "'(404) 727-7611', "
42:                + "'carterweb@emory.edu')");
43:            result = st.executeUpdate(
44:                "INSERT INTO contacts (name, address1, address2, "
45:                + "phone, email) VALUES("
46:                + "'George Bush', "
47:                + "'Box 79798', "
48:                + "'Houston, TX 77279', "
49:                + "'(409) 260-9552', "
50:                + "'library@bush.nara.gov')");
51:            result = st.executeUpdate(
52:                "INSERT INTO contacts (name, address1, address2, "
53:                + "phone, email) VALUES("
54:                + "'Bill Clinton', "
55:                + "'15 Old House Lane', "
56:                + "'Chappaqua, NY 10514', "
57:                + "'(501) 370-8000', "
58:                + "'info@clintonpresidentialcenter.com')");
59:            result = st.executeUpdate(
60:                "INSERT INTO contacts (name, address1, address2, "
61:                + "phone, email) VALUES("
62:                + "'George W. Bush', "
63:                + "'White House, 1600 Pennsylvania Ave.', "
64:                + "'Washington, DC 20500', "
65:                + "'(202) 456-1414', "
66:                + "'president@whitehouse.gov')");
67:            st.close();
68:            System.out.println("Database created in " + system);
69:        } catch (Exception e) {
70:            System.out.println("Error - " + e.toString());

```

```

71:     }
72: }
73:
74: public void readDatabase() {
75:     String data = "jdbc:derby:presidents";
76:     try {
77:         // load the driver and connect to the database
78:         Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
79:         Connection conn = DriverManager.getConnection(
80:             data, "", "");
81:         // load all records from the contacts table
82:         Statement st = conn.createStatement();
83:         ResultSet rec = st.executeQuery(
84:             "SELECT * FROM contacts ORDER BY name");
85:         // loop through each record and display its fields
86:         while(rec.next()) {
87:             System.out.println(rec.getString("name") + "\n"
88:                 + rec.getString("address1") + "\n"
89:                 + rec.getString("address2") + "\n"
90:                 + rec.getString("phone") + "\n"
91:                 + rec.getString("email") + "\n");
92:         }
93:         st.close();
94:     } catch (Exception e) {
95:         System.out.println("Error - " + e.toString());
96:     }
97: }
98:
99: public static void main(String[] arguments) {
100:     Presidents prez = new Presidents();
101:     if (arguments.length < 1) {
102:         System.out.println("Usage: java Presidents {create|read}");
103:         System.exit(-1);
104:     }
105:     if (arguments[0].equals("create")) {
106:         prez.createDatabase();
107:     }
108:     if (arguments[0].equals("read")) {
109:         prez.readDatabase();
110:     }
111: }
112: }

```

要在该应用程序中使用其他数据库和驱动程序，需要修改第 16、19、75 和 77 行。

Java DB 要求将系统属性 `derby.system.home` 设置为数据库所处的根目录。如果该目录不存在，Java DB 将创建它。

可使用下述语句加载 Java DB 的 JDBC 驱动程序：

```
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
```

应用程序 `Presidents` 包含方法 `createDatabase()` 和 `readDatabase()`，这两个方法的作用是不言自明的。

在第 21 行使用 `DriverManager.getConnection(String)` 创建了数据库，指定的数据库连接字符串如下：

```
jdbc:derby:presidents;create=true
```

这个字符串以 `jdbc:derby:` 打头，接下来是数据库名称、分号和参数 `create=true`，该参数（导致）必要时将创建该数据库。

在该字符串中，也可包含参数 `user` 和 `password`，用于登录数据库：

```
jdbc:derby:presidents;user=dbuser;password=tortuga;create=true
```

第 79 行建立了一条连接以便读取数据库中的信息：

```
jdbc:derby:presidents
```

连接到 Java DB 后，通过 JDBC 读写数据库记录的方式与通过 JDBC-ODBC 读写数据相同：编写 SQL 语句来创建数据库表、将记录插入到表中以及读取这些记录。

Java DB 使用的 SQL 支持的记录类型不同于 Access、MySQL 和其他数据库。

首次运行应用程序 `Presidents` 时，将 `create` 作为唯一的参数，这将创建一个新的数据库：

```
java Presidents create
```

如果成功，该应用程序将显示类似于下面的消息：

```
Database created in C:\Documents and Settings\Rogers\.database
```

再次运行该应用程序，并将 read 作为参数以读取并显示数据库的内容：

```
java Presidents read
```

该应用程序的输出如下：

```
Bill Clinton
15 Old House Lane
Chappaqua, NY 10514
(501) 370-8000
info@clintonpresidentialcenter.com
```

```
George Bush
Box 79798
Houston, TX 77279
(409) 260-9552
library@bush.nara.gov
```

```
George W. Bush
White House, 1600 Pennsylvania Ave.
Washington, DC 20500
(202) 456-1414
president@whitehouse.gov
```

```
Jimmy Carter
Carter Presidential Center
1 Copenhill, Atlanta, GA 30307
(404) 727-7611
carterweb@emory.edu
```

提供 Java DB 是 Java 6 最重要的改进之一。这样，所有安装了 Java 的计算机都将有一个关系型数据库，这让程序员能够利用永久性数据存储。

## 18.3 总结

本章介绍了如何操纵以诸如 Microsoft Access 和 Java DB 等流行的数据库格式存储的数据。通过使用 JDBC 或结合使用 JDBC 和 ODBC，可以在 Java 程序中使用现有的数据存储解决方案。

通过使用 JDBC/ODBC 和结构化查询语言 (SQL，这是一种读写和管理数据库的标准语言)，可以连接到多种关系型数据库，并对数据库进行管理。

## 18.4 问与答

问：JDBC-ODBC 桥驱动程序可用于小程序中吗？

答：小程序的默认安全策略不允许使用 JDBC-ODBC 桥，因为其 ODBC 端使用的是本机代码而不是 Java。不能对本机代码实施 Java 的安全限制，因此无法确保这种代码是安全的。

完全用 Java 实现的 JDBC 驱动程序可用于小程序中，它有这样的优点，即不要求在客户机上做任何配置。

问：Java DB 同诸如 Access 和 MySQL 等著名数据库之间有何不同？应使用哪种数据库？

答：Java DB 是为需求简单的数据库应用程序提供的。整个 Java DB 最多可占用 2MB 空间，这使其更容易同需要连接数据库的 Java 应用程序捆绑。

Sun 公司在 Java Enterprise Edition 的多个地方都使用了 Java DB，这表明它在执行重要任务时能够提供稳定、可靠的性能。



## 18.5 小测验

请回答下述问题，以复习本章介绍的内容。

### 18.5.1 问题

1. 在数据库程序中，Statement 对象表示什么？
  - a. 到数据库的连接；
  - b. 用 SQL 编写的数据库查询；
  - c. 数据源。
2. 哪个 Java 类用于表示执行前已编译的 SQL 语句？
  - a. Statement；
  - b. PreparedStatement；
  - c. ResultSet。
3. 方法 Class.forName(String)有何功能？
  - a. 提供类的名称；
  - b. 加载可用于访问数据库的数据库驱动程序；
  - c. 删除一个对象。

答案

1. b，这个类位于 java.sql 包中，表示一条 SQL 语句。
2. b，由于被编译，因此对于要多次执行的 SQL 查询，PreparedStatement 是更合适的选择。
3. b，这个静态方法加载数据库驱动程序。

### 18.5.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

对于下述代码：

```
public class ArrayClass {

    public static ArrayClass newInstance() {
        count++;
        return new ArrayClass();
    }

    public static void main(String arguments[]) {
        new ArrayClass();
    }

    int count = -1;
}
```

哪行代码将导致该程序无法通过编译：

- a. count++;
- b. return new ArrayClass();
- c. public static void main(String arguments[]) {
- d. int count = -1;

答案 a



## 18.6 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 修改应用程序 CoalReporter，以读取表 Country Oil Totals 而不是 Coal 中的字段。
2. 编写一个应用程序，它将 Yahoo! 的股票报价存储到一个 Java DB 数据库中。



## 第 19 章

# 读写 RSS Feed

在本章的课程中，您将使用可扩展标记语言（XML），这是一种格式标准，让数据完全是可移植的。将从如下方面介绍 XML：

- 将数据表示为 XML 格式；
- 为何 XML 是一种有效的数据存储方式；
- 使用 XML 发布 Web 内容；
- 读写 XML 数据。

本章将使用的 XML 格式是 Really Simple Syndication（RSS），这是一种发布 Web 内容和共享信息的流行方式，被众多网站采用。

### 19.1 使用 XML

Java 的主要卖点之一是，使用它可以编写出能够运行在不同操作系统中的程序，而无需做任何修改。在当前的计算环境中，Windows、Linux、Mac OS 以及其他几个操作系统被广泛使用，而很多人使用多种操作系统，因此软件的可移植性提供了极大的便利。

XML 指的是可扩展标记语言（Extensible Markup Language），它是一种用于存储和组织数据的格式，独立于处理数据的软件程序。

XML 格式的数据易于被重用，原因有以下几个。

首先，数据是以标准方式组织的，这样，支持 XML 的软件程序都能够读写这种数据。如果您创建了一个 XML 文件，用于表示公司的雇员数据库，则几十个 XML 分析程序都能够读取该文件，并理解其内容。

不管您收集的雇员信息是什么，情况都将如此。如果数据库中只包含雇员的姓名、ID 号和薪水，XML 分析程序能够读取它；如果它包含 25 个不同的数据项，如生日、血型 and 头发颜色等，分析程序也能够读取它。

其次，数据对自身做了说明，这使得人们只需使用文本编辑器对文件进行查看，就能知道其用途。只要打开您的 XML 雇员信息数据库，任何人都能够知道每条雇员记录的结构和内容，而不需要您的帮助。

程序清单 19.1 说明了这一点。该程序清单包含一个 RSS 文件。由于 RSS 是一种 XML 方言，因此其结构符合 XML 规则。

程序清单 19.1 完整的 workbench.rss 源代码

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <rss version="2.0">
3:   <channel>
4:     <title>Workbench</title>
```

```

5:    <link>http://www.cadenhead.org/workbench/</link>
6:    <description>Programming, publishing, politics, and popes</description>
7:    <docs>http://www.rssboard.org/rss-specification</docs>
8:    <item>
9:      <title>Toronto Star: Only 100 Blogs Make Money</title>
10:     <link>http://www.cadenhead.org/workbench/news/3132</link>
11:     <pubDate>Mon, 26 Feb 2007 11:30:57 -0500</pubDate>
12:     <guid isPermaLink="false">tag:cadenhead.org,2007:weblog.3132</guid>
13:     <enclosure length="2498623" type="audio/mpeg"
14:       url="http://mp3.cadenhead.org/3132.mp3" />
15:   </item>
16:   <item>
17:     <title>Eliot Spitzer Files UDRP to Take EliotSpitzer.Com</title>
18:     <link>http://www.cadenhead.org/workbench/news/3130</link>
19:     <pubDate>Thu, 22 Feb 2007 18:02:53 -0500</pubDate>
20:     <guid isPermaLink="false">tag:cadenhead.org,2007:weblog.3130</guid>
21:   </item>
22:   <item>
23:     <title>Fuzzy Zoeller Sues Over Libelous Wikipedia Page</title>
24:     <link>http://www.cadenhead.org/workbench/news/3129</link>
25:     <pubDate>Thu, 22 Feb 2007 13:48:45 -0500</pubDate>
26:     <guid isPermaLink="false">tag:cadenhead.org,2007:weblog.3129</guid>
27:   </item>
28: </channel>
29: </rss>

```

请在字处理器或文本编辑器中输入上述代码，并将其保存为纯文本文件 workbench.rss。

您知道这些数据表示的是什么呢？虽然开头的?xml 标记可能不太好懂，但其他内容显然是一个某种网站数据库。

第 1 行的?xml 标记有一个 version 属性，其值为 1.0，还有一个值为 utf-8 的 encoding 属性。这表明该文件遵循 XML 1.0 的规则，使用的字符集为 UTF-8。

XML 中的数据位于标记元素之间，而标记元素对数据进行了描述。开始标记以<打头，然后是标记的名称和>；结束标记以</打头，然后是标记名和>。例如，在程序清单 19.1 中，第 8 行的<item>是开始标记，而第 15 行的</item>是结束标记。这两个标记之间的所有内容都被视为元素的值。

元素可嵌套在其他元素中，从而形成 XML 数据的层次结构，这种结构指定了数据之间的关系。在程序清单 19.1 中，第 9~14 行的内容都是相关的，其中的每个元素都定义了同一项网站内容的某方面。

元素也可以包含属性，属性由数据构成，对与标记相关联的数据进行补充。属性是在开始标记中定义的。属性名后是等号和用引号括起的文本。

在程序清单 19.1 的第 12 行，元素 guid 包含一个值为 false 的 isPermaLink 属性，这表明该元素的值 tag:cadenhead.org,2007:weblog.3132 不是 permalink，后者指的是可加载到浏览器中的内容的 URL。

XML 还支持使用单个（而不是一对）标记来定义元素。在这种情况下，使用的标记以<打头，然后是标记名，再以字符/>结束。在上述 RSS 文件中，第 13~14 行是一个 enclosure 元素，它描述了一个相关联的 MP3 音频文件。

XML 鼓励创建易于理解和使用的数据，即使用户没有创建这些数据程序且找不到任何描述它的文档。

对于程序清单 19.1 中的 RSS 文件，在很大程度上说，只需通过查看就能知道其用途。其中的每项内容 (item) 表示一个最近更新过的网页。

#### 提示

当前，通过 RSS 和一种类似的格式——Atom——发布新的网站内容已成为通过网络建立读者关系的最佳方式之一。很多人都使用诸如 Google Reader、Bloglines 和 My Yahoo 等阅读器软件订阅 RSS 文件，这种文件称为 feed。

本书的第一作者 Rogers Cadenhead 当前是 RSS 顾问委员会的主席，该委员会是发布 RSS 2.0 规范的组。有关这种格式的更详细信息，请访问该委员会的网站 <http://www.rssboard.org>，也可在 <http://www.rssboard.org/rss-feed> 订阅其 RSS feed。

遵循了 XML 格式规则的数据称为格式良好 (well-form) 的，任何支持 XML 的软件都能够读写格式良好的 XML 数据。

#### 注意

通过支持格式良好的标记，XML 简化了编写处理数据的程序的任务。RSS 使得能够以软件易于处理的格式提供网站更新。本书的作者之一在 <http://www.cadenhead.org/workbench> 发布了针对 Workbench 的 RSS feed，这可供两种受众使用：人类和计算机，其中前者使用其喜欢的 RSS 阅读器阅读博客，而后者（如 Technorati）对数据进行处理。Technorati 提供了可搜索的网站更新数据库、不同博客之间的连接以及分类。有关 Technorati 如何使用该 RSS feed，请访问 <http://technorati.com/blogs/cadenhead.org/workbench>。

## 19.2 设计 XML 语言

虽然人们将 XML 视为一种语言，并将其与超文本标记语言 (HTML) 相提并论，但其范畴要大得多。XML 是一种标记语言，它定义了如何定义标记语言。

这种说法很古怪，类似于哲学书上的内容。然而理解这种概念非常重要，因为它解释了如何使用 XML 来定义各种数据——卫生保健知识、家谱、报刊文章和分子等。

XML 中的“X”表示 Extensible (可扩展的)，它指的是根据自己的目的来组织数据。使用 XML 的规则组织起来的数据可以表示任何东西：

- 电话销售公司的程序员可使用 XML 来存储外拨电话、拨出时间、电话号码、打电话的人以及结果；
- 业余爱好者可使用 XML 来记录收到的推销电话、电话的时间、打电话的公司以及推销的产品；
- 政府机构的程序员可使用 XML 来记录有关电话推销的投诉——推销公司的名称和被投诉次数。

上述例子都使用 XML 来定义一种能够满足特定目的的新语言。您可以将其称作 XML 语言，或 XML 文档类型。

可使用文档类型定义 (DTD) 来设计 XML 语言，DTD 指出了其涵盖的潜在元素和属性。

在 XML 文件中，可在开头的 ?xml 后面放置特殊的 !DOCTYPE 声明，用于指出该文件的 DTD，如下所示：

```
<!DOCTYPE Library SYSTEM "librml.dtd">
```

声明 !DOCTYPE 用于标识用于数据的 DTD。有了 DTD 后，很多 XML 工具就可以读取根据该 DTD 创建的 XML，并判断数据是否遵循了所有的规则。如果没有，XML 工具将拒绝它，并指出导致错误的代码行。这被称为验证 XML。

使用 XML 时，您将遇到的一种情况是，数据被作为 XML 组织起来，但没有使用 DTD 对其进行定义。大多数 RSS 文件都不需要指定 DTD。这些数据可被分析（如果格式良好），因此您可以将其读入到程序中，并对其进行处理，但您无法检查其有效性，以确保它按相应语言的规则被正确地组织起来。

#### 提示

要了解人们开发了哪些类型的 XML 语言，请参阅 Cover Pages 在 <http://xml.coverpages.org/xmlApplications.html> 提供的清单。

## 19.3 使用 Java 处理 XML

Java 通过用于处理 XML 的 Java API 支持 XML，这是一组用于读写和操纵 XML 数据的 Java 类。

`javax.xml.parsers` 包是其他包的入口。这些类可用于分析和验证 XML 数据，这需要使用两种技术：Simple API for XML (SAX) 和文档对象模型 (Document Object Model, DOM)。然而，它们难以实现，这使得其他组织提供了用于处理 XML 的类库。

在本章余下的内容中，将使用其中的一个类库：XML 对象模型 (XOM) 类库，这是一个开源 Java 类库，使得读写和转换 XML 数据都非常简单。

### 注意

有关用于处理 XML 的 Java API 的更详细信息，请访问 Sun 公司的 Java 网站，网址为 <http://java.sun.com/xml/>。

## 19.4 使用 XOM 处理 XML

作为 Java 程序员，最重要的技能之一是找到适合在自己的项目中使用的包和类。重用设计良好的类库比自己开发类更容易，其原因是显而易见的。

虽然 Sun 公司提供的 Java 类库包含数千个设计良好的类，能够满足多种开发需求，但并非只有该公司提供对您的工作有帮助的包。

其他公司、团体和个人以商业和开源许可的方式提供了数十个 Java 包。其中一些最著名的包来自 Apache Jakarta，它是 Apache 软件基金会的一个 Java 开发项目，开发了 Web 应用程序框架 Struts、日志类库 Log4J 和众多的其他类库。

另一个了不起的开源 Java 类库是 XOM 库，它是一个基于树 (tree-based) 的 XML 处理包，其宗旨是易于学习和使用，同时遵循格式良好的 XML。

这个库是程序员兼作者 Elliott Rusty Harold，根据自己使用 Sun XML 处理包的经验以及使用 Java 处理 XML 的心得体会开发出来的。

这个项目最初是 JDOM 的一部分。JDOM 是一个基于树的、表示 XML 文档的模型。Harold 为这个开源项目编写代码，并参与了开发工作。

Harold 没有继承 JDOM 代码，而是决定从空白开始，并在 XOM 中修改了 JDOM 的一个核心设计原则。

这个库采用了如下原则：

- 将 XML 文档视为一棵树，并用 Java 类来表示树中的节点，如元素、注释、处理指令和文档类型声明等。程序员可以添加和删除节点，从而对内存中的文档进行操纵，这种简单的方法可以使用 Java 来妥善地实现。
- XOM 生成的所有 XML 数据都是格式良好的，并有格式良好的名称空间。
- 对于 XML 文档中的每个元素，都用一个包含构造函数的类表示。
- 不支持对象串行化，而是提倡程序员将 XML 用作串行化数据的格式，这样将能够与任何能够读取 XML 的软件交换这种数据，无论这些软件是使用哪种编程语言开发的。
- 这个库依赖于另一个 XML 分析程序来读取 XML 文档和填充树，而不是自己直接去完成低级工作。XOM 使用一个 SAX 分析程序，该程序必须单独下载和安装。当前，首选分析程序是 Apache Xerces 2.7.1。

下载 XOM 和 Xerces，并将其包加入到系统的 Classpath 变量中后，便可以使用 XOM。

完整的安装说明请参阅 XOM 和 Xerces 网站。这些类是以 JAR 存档文件 xom-1.1.jar、xercesImpl.jar 和 xml-apis.jar 分发的。应将这些文件加入到系统的环境变量 Classpath 中，这样 Java 程序才能使用这些 XOM 类。

### 19.4.1 创建 XML 文档

接下来要创建的应用程序 RssStarter 创建一个 XML 文档，该文档包含一个 RSS feed 开头的内容，如程序清单 19.2 所示。

程序清单 19.2 feed.rss 的完整内容

---

```

1: <?xml version="1.0"?>
2: <rss version="2.0">
3:   <channel>
4:     <title>Workbench</title>
5:     <link>http://www.cadenhead.org/workbench/</link>
6:   </channel>
7: </rss>

```

---

nu.xom 包中包含表示 XML 文档的类 (Document) 和表示文档中节点的类 (Attribute、Comment、DocType、Element、ProcessingInstruction 和 Text)。

应用程序 RssStarter 使用了其中的几个类。首先通过将元素的名称指定为参数来创建了 Element 对象：

```
Element rss = new Element("rss");
```

上述语句创建一个用作文档根元素的对象：rss。可以使用 Element 的只接受一个参数的构造函数，因为该文档没有使用一种名为名称空间的 XML 特性。如果使用了这种特性，必须指定第二个参数：名称空间统一资源定位符 (URI)。XOM 库中的其他类以类似的方式支持名称空间。

在程序清单 19.2 中的 XML 文档中，元素 rss 包括一个名为 version 的属性，该属性的值为 2.0。属性可通过指定其名称和值来创建：

```
Attribute version = new Attribute("version", "2.0");
```

要将属性加入到元素中，可调用元素的方法 addAttribute()，并将属性作为唯一的参数：

```
rss.addAttribute(version);
```

元素中的文本用 Text 对象表示，这种对象可通过将文本指定为 String 参数来创建：

```
Text titleText = new Text("Workbench");
```

在 XML 文档中，所有元素都位于根元素内部，后者用于创建一个 Document 对象。创建 Document 对象的方法是，调用构造函数 Document，并将根元素作为参数。在应用程序 RssStarter 中，根元素名为 rss。任何 Element 对象都可用作文档的根：

```
Document doc = new Document(rss);
```

在 XOM 的树型结构中，表示 XML 文档及其组成部分的类被组织成层次结构，其中最顶层是通用超类 nu.xom.Node。这个类有 3 个子类：Attribute、LeafNode 和 ParentNode，它们位于同一个包中。

要将子节点加入到父节点中，可调用父节点的 appendChild() 方法，并将要加入的节点作为参数。下面的代码创建 3 个元素——一个名为 domain 的父节点以及两个分别名为 name 和 dns 的子节点：

```

Element channel = new Element("channel");
Element link = new Element("link");
Text linkText = new Text("http://www.cadenhead.org/workbench/");
link.appendChild(linkText);
channel.appendChild(link);

```

方法 `appendChild()` 将一个新的子节点附加到父节点的其他所有子节点的后面。上述语句生成如下 XML 文档片段：

```

<channel>
  <link>http://www.cadenhead.org/workbench/</link>
</channel>

```

调用方法 `appendChild()` 时，也可以指定一个 `String` 参数，而不是将节点作为参数。

`Text` 对象表示要加入到元素中的字符串：

```
link.appendChild("http://www.cadenhead.org/workbench/");
```

创建好树并加入节点后，便可以调用 `Document` 的方法 `toXML()` 来显示它。这个方法返回一个字符串，其中包含格式良好的整个 XML 文档。

程序清单 19.3 是该应用程序的源代码。

#### 程序清单 19.3 完整的 RssStarter.java 源代码

```

1: import nu.xom.*;
2:
3: public class RssStarter {
4:     public static void main(String[] arguments) {
5:         // create an <rss> element to serve as the document's root
6:         Element rss = new Element("rss");
7:
8:         // add a version attribute to the element
9:         Attribute version = new Attribute("version", "2.0");
10:        rss.addAttribute(version);
11:        // create a <channel> element and make it a child of <rss>
12:        Element channel = new Element("channel");
13:        rss.appendChild(channel);
14:        // create the channel's <title>
15:        Element title = new Element("title");
16:        Text titleText = new Text("Workbench");
17:        title.appendChild(titleText);
18:        channel.appendChild(title);
19:        // create the channel's <link>
20:        Element link = new Element("link");
21:        Text linkText = new Text("http://www.cadenhead.org/workbench/");
22:        link.appendChild(linkText);
23:        channel.appendChild(link);
24:
25:        // create a new document with <rss> as the root element
26:        Document doc = new Document(rss);
27:
28:        // Display the XML document
29:        System.out.println(doc.toXML());
30:    }
31: }

```

应用程序 `RssStarter` 将其创建的 XML 文档显示到标准输出中。下面的命令运行该应用程序，并将输出重定向到一个名为 `feed.rss` 的文件中：

```
java RssStarter > feed.rss
```

XOM 自动在文档开头加上 XML 声明。

该应用程序生成的 XML 文档没有采用缩进格式：所有元素都放在一行。

表示 XML 数据时，XOM 只保留必不可少的空格——程序清单 19.2 所示的 RSS feed 包含的空格



只是为了方便演示，XOM 创建 XML 文档时，并不会自动生成这些空格。接下来的示例演示如何控制缩进格式。

## 19.4.2 修改 MXL 文档

下一个应用程序 DemainEditor 对应用程序 RssStarter 生成的 XML 文档 (feed.rss) 做了几处修改。将元素 link 中的文本改为 <http://www.cadenhead.org/>，还新增了一个 item 元素：

```
<item>
  <title>Fuzzy Zoeller Sues Over Libelous Wikipedia Page</title>
</item>
```

使用 nu.xom 包，可以将 XML 文档加载到树中。这些文档可以位于多个地方：File、InputStream、Reader 或 URL。URL 是用 String 而不是 java.net.URL 对象指定的。

Builder 类表示 SAX 分析程序，能够将 XML 文档加载到 Document 对象中。可以在调用构造函数时指定要使用的分析程序。如果没有指定，XOM 将按下列顺序使用第一个可用的分析程序：Xerces 2、Piccolo、GNU Aelfred、Oracle、XP、Saxon Aelfred 和 Dom4J Aelfred。如果没有找到上述任何一个分析程序，将使用系统属性 org.xml.sax.driver 指定的分析程序。构造函数还决定了分析程序是否进行有效性检查。

构造函数 Builder() 和 Builder(true) 都使用默认分析程序：很可能是某个版本的 Xerces。在上述第二个构造函数中，布尔参数 true 命令分析程序对 XML 文档进行有效性检查。如果没有指定该参数，将不进行有效性检查。如果根据文档类型定义的规则，XML 文档不能通过有效性检查，分析程序将引发 nu.xom.ValidityException 异常。

Builder 对象的 build() 方法加载 XML 文档，并返回一个 Document 对象：

```
Builder builder = new Builder();
File xmlFile = new File("feed.rss");
Document doc = builder.build(xmlFile);
```

上述语句加载文件 feed.rss 中的 XML 文档，这可能出现下列两个问题之一：如果文件中没有包含格式良好的 XML 文档，将引发 nu.xom.ParseErrorException 异常；如果输入操作失败，将引发 java.io.IOException 异常。

要检索树中的元素，可调用其父节点的相应方法。

Document 对象的 getRootElement() 方法返回文档的根元素：

```
Element root = doc.getRootElement();
```

在 XML 文档 feed.rss 中，根元素是 rss。

要检索有名称的元素，可调用其父节点的 getChildElement() 方法，并将元素的名称作为方法的 String 参数：

```
Element channel = root.getChildElement("channel");
```

上述语句取回元素 rss 中的元素 channel，如果没有找到这样的元素，将返回 null。和其他示例一样，这里出于简化的目的，在文档中没有使用名称空间，也有将元素名称和名称空间作为参数的方法。

如果同一个父节点中有多个名称相同的元素，可以使用父节点的 getChildElements() 方法来取回这些元素：

```
Elements children = channel.getChildElements()
```

getChildElements() 方法返回一个 Elements 对象，其中包含所有符合条件的元素。这个对象是一个

只读的链表。如果调用 `getChildElements()` 后，父节点的内容发生了变化，这个对象的内容不会自动变化。

`Elements` 有一个 `size()` 方法，该方法返回一个整数，指出 `Elements` 中包含多少个元素。可以在循环中使用这个方法遍历所有的元素，元素编号从 0 开始。方法 `get()` 取回 `Elements` 中的一个元素，它接受一个整型参数，该参数指出了要取回的元素的位置：

```
for (int i = 0; i < children.size(); i++) {    Element link = children.get(i);
}
```

上述 for 循环遍历根元素 `channel` 的所有子元素。

要取回没有名称的元素，可调用其父节点的 `getChild()` 方法，并指定一个参数：一个指出元素在父节点中位置的整数：

```
Text linkText = (Text) link.getChild(0);
```

上述语句创建一个 `Text` 对象，用于存储元素 `link` 中的文本 `http://www.cadenhead.org/workbench`。在其父元素中，`Text` 元素的位置总是 0。

要将上述文本作为 `String` 进行处理，可调用 `Text` 对象的 `getValue()` 方法，如下述语句所示：

```
if (linkText.getValue().equals("http://www.cadenhead.org/workbench/"))
    // ...
}
```

仅当 `link` 元素包含文本 `http://www.cadenhead.org/workbench/` 时，应用程序 `DomainEditor` 才对其进行修改。该应用程序对文档做如下修改：删除 `link` 元素的文本，并在原来的位置加入新文本 `http://www.cadenhead.org/`，然后新添一个 `item` 元素。

父节点有两个 `removeChild()` 方法，可用于将子节点从文档中删除。使用一个整型参数调用该方法时，将删除相应位置的子节点：

```
Element channel = domain.getFirstChildElement("channel");
Element link = dns.getFirstChildElement("link");
link.removeChild(0);
```

上述语句删除 `channel` 中第一个 `link` 元素的 `Text` 对象。

使用节点作为参数调用 `removeChild()` 方法时，将删除指定的节点。例如，要删除 `link` 元素，可使用如下语句：

```
channel.removeChild(link);
```

程序清单 19.4 列出了应用程序 `DomainEditor` 的源代码。

#### 程序清单 19.4 DomainEditor.java 的完整源代码

```
1: import java.io.*;
2: import nu.xom.*;
3:
4: public class DomainEditor {
5:     public static void main(String[] arguments) throws IOException {
6:         try {
7:             // create a tree from the XML document feed.rss
8:             Builder builder = new Builder();
9:             File xmlFile = new File("feed.rss");
10:            Document doc = builder.build(xmlFile);
11:
12:            // get the root element <rss>
13:            Element root = doc.getRootElement();
14:
15:            // get its <channel> element
```

```

16:         Element channel = root.getFirstChildElement("channel");
17:
18:         // get its <link> elements
19:         Elements children = channel.getChildElements();
20:         for (int i = 0; i < children.size(); i++) {
21:
22:             // get a <link> element
23:             Element link = children.get(i);
24:
25:             // get its text
26:             Text linkText = (Text) link.getChild(0);
27:
28:             // update any link matching a URL
29:             if (linkText.getValue().equals(
30:                 "http://www.cadenhead.org/workbench/")) {
31:
32:                 // update the link's text
33:                 link.removeChild(0);
34:                 link.appendChild("http://www.cadenhead.org/");
35:             }
36:         }
37:
38:         // create new elements and attributes to add
39:         Element item = new Element("item");
40:         Element itemTitle = new Element("title");
41:
42:         // add them to the <channel> element
43:         itemTitle.appendChild(
44:             "Fuzzy Zoeller Sues Over Libelous Wikipedia Page"
45:         );
46:         item.appendChild(itemTitle);
47:         channel.appendChild(item);
48:
49:         // display the XML document
50:         System.out.println(doc.toXML());
51:     } catch (ParsingException pe) {
52:         System.out.println("Error parsing document: " + pe.getMessage());
53:         pe.printStackTrace();
54:         System.exit(-1);
55:     }
56: }
57: }

```

应用程序 DomainEditor 将修改后的 XML 文档显示到标准输出中。可以使用下述命令来运行该应用程序，以生成一个名为 feeds2.rss 的文件：

```
java DomainEditor > feed2.rss
```

### 19.4.3 格式化 XML 文档

正如前面指出的，表示 XML 文档时，XOM 不会保留没有意义的空格。这符合 XOM 的设计目标之一：不考虑 XML 文档中任何没有语法意义的内容。另一个例子是，不管文本是使用字符实体、CDATA 部分还是常规字符创建的，处理的方式都一样。

下一个应用程序 DomainWriter 在 XML 文档 feeds2.rss 的开头添加注释，并采用缩进格式将文档内容排列成多行，从而生成程序清单 19.5 所示的版本。

程序清单 19.5 完整的 feeds2.rss

```

1: <?xml version="1.0"?>
2: <rss version="2.0">
3:   <channel>
4:     <title>Workbench</title>
5:     <link>http://www.cadenhead.org/</link>
6:     <item>
7:       <title>Fuzzy Zoeller Sues Over Libelous Wikipedia Page</title>

```

```

8:      </item>
9:    </channel>
10: </rss>

```

nu.xom 包中的 `Serializer` 类让您能够在显示或存储 XML 文档时控制其格式化方式。缩进、字符编码方法、换行符以及其他格式都是使用这种类的对象指定的。

要创建 `Serializer` 对象，可调用其构造函数，并将输出流和字符编码方法作为参数：

```

File inFile = new File(arguments[0]);
FileOutputStream fos = new FileOutputStream("new_" +
    inFile.getName());
Serializer output = new Serializer(fos, "ISO-8859-1");

```

上述语句使用字符编码方法 ISO-8859-1 串行化一个文件，文件名是使用一个命令行参数指定的。

当前，`Serializer` 支持 20 种编码方法，其中包括 ISO-10646-UCS-2、ISO-8859-1 至 ISO-8859-10、ISO-8859-13 至 ISO-8859-16、UTF-8 和 UTF-16。还有一个只接受输出流作为参数的 `Serializer()` 构造函数，这样默认将使用编码方法 UTF-8。

要设置缩进，可调用 `Serializer` 的 `setIndentation()` 方法，并指定一个整型参数，它指出了空格数：

```
output.setIndentation(2);
```

要将整个 XML 文档写入到 `Serializer` 的目的地，可调用其 `write()` 方法，并将文档作为参数：

```
output.write(doc);
```

应用程序 `DomainWriter` 将注释插入到 XML 文档的开头，而不是将其放在父节点的子节点后面。

为此，需要使用父节点的另一个方法 `insertChild()`，该方法接受两个参数——要加入的元素和插入位置：

```

Builder builder = new Builder();
Document doc = builder.build(arguments[0]);
Comment timestamp = new Comment("File created " +
    new java.util.Date());
doc.insertChild(timestamp, 0);

```

注释被放在文档开头，这使得 `doamins` 标记下移了一行，但仍位于 XML 声明的后面。

程序清单 19.6 列出了该应用程序的源代码。

#### 程序清单 19.6 完整的 `DomainWriter.java` 源代码

```

1: import java.io.*;
2: import nu.xom.*;
3:
4: public class DomainWriter {
5:     public static void main(String[] arguments) throws IOException {
6:         try {
7:             // Create a tree from an XML document
8:             // specified as a command-line argument
9:             Builder builder = new Builder();
10:            Document doc = builder.build(arguments[0]);
11:
12:            // Create a comment with the current time and date
13:            Comment timestamp = new Comment("File created "
14:                + new java.util.Date());
15:
16:            // Add the comment above everything else in the
17:            // document
18:            doc.insertChild(timestamp, 0);
19:
20:            // Create a file output stream to a new file
21:            File inFile = new File(arguments[0]);
22:            FileOutputStream fos = new FileOutputStream("new_" +
23:                inFile.getName());
24:
25:            // Using a serializer with indentation set to 2 spaces,
26:            // write the XML document to the file
27:            Serializer output = new Serializer(fos, "ISO-8859-1");
28:            output.setIndent(2);
29:            output.write(doc);
30:        } catch (ParsingException pe) {
31:            System.out.println("Error parsing document: " + pe.getMessage());

```

```

31:         pe.printStackTrace();
32:         System.exit(-1);
33:     }
34: }
35: }

```

应用程序 DomainWriter 运行时，接受一个 XML 文件名作为命令行参数：

```
java DomainWriter feeds2.rss
```

上述命令生成一个名为 new\_feeds2.rss 的文件，它是前述 XML 文档的缩进版本，且包含作为注释的时间戳。

### 19.4.4 评估 XOM

前述 3 个应用程序涵盖了 XOM 包的核心特性，它们表明 XOM 包处理 XML 文档的方法相当简单。

另外还有一些小型包：nu.xom.canonical、nu.xom.converters、nu.xom.xinclude 和 nu.xom.xslt，用于支持 XInclude、XSLT、规范 XML 串行化以及 XOM 模型与 DOM 和 SAX 使用的模型之间的转换。

程序清单 19.7 中的应用程序使用来自动态源的 XML 文档：RSS 提供的最新 Web 内容。应用程序 RssFilter 在新闻标题中查找指定文本，然后生成一个新的 XML 文档，其中只包含符合条件的内容，并采用缩进格式。它还修改了标题，并在必要时添加 RSS 0.91 文档类型声明。

程序清单 19.7 完整的 RssFilter.java 源代码

```

1: import nu.xom.*;
2:
3: public class RssFilter {
4:     public static void main(String[] arguments) {
5:
6:         if (arguments.length < 2) {
7:             System.out.println("Usage: java RssFilter rssFile searchTerm");
8:             System.exit(-1);
9:         }
10:
11:         // Save the RSS location and search term
12:         String rssFile = arguments[0];
13:         String searchTerm = arguments[1];
14:
15:         try {
16:             // Fill a tree with an RSS file's XML data
17:             // The file can be local or something on the
18:             // Web accessible via a URL.
19:             Builder bob = new Builder();
20:             Document doc = bob.build(rssFile);
21:
22:             // Get the file's root element (<rss>)
23:             Element rss = doc.getRootElement();
24:
25:             // Get the element's version attribute
26:             Attribute rssVersion = rss.getAttribute("version");
27:             String version = rssVersion.getValue();
28:
29:             // Add the DTD for RSS 0.91 feeds, if needed
30:             if ( (version.equals("0.91")) & (doc.getDocType() == null) ) {
31:                 DocType rssDtd = new DocType("rss",
32:                     "http://my.netscape.com/publish/formats/rss-0.91.dtd");
33:                 doc.insertChild(rssDtd, 0);
34:             }
35:
36:             // Get the first (and only) <channel> element
37:             Element channel = rss.getFirstChildElement("channel");
38:
39:             // Get its <title> element
40:             Element title = channel.getFirstChildElement("title");
41:             Text titleText = (Text)title.getChild(0);

```

```

42:
43:         // Change the title to reflect the search term
44:         titleText.setValue(titleText.getValue() + ": Search for " +
45:             searchTerm + " articles");
46:
47:         // Get all of the <item> elements and loop through them
48:         Elements items = channel.getChildElements("item");
49:         for (int i = 0; i < items.size(); i++) {
50:             // Get an <item> element
51:             Element item = items.get(i);
52:
53:             // Look for a <title> element inside it
54:             Element itemTitle = item.getFirstChildElement("title");
55:
56:             // If found, look for its contents
57:             if (itemTitle != null) {
58:                 Text itemTitleText = (Text) itemTitle.getChild(0);
59:
60:                 // If the search text is not found in the item,
61:                 // delete it from the tree
62:                 if (itemTitleText.toString().indexOf(searchTerm) == -1)
63:                     channel.removeChild(item);
64:             }
65:         }
66:
67:         // Display the results with a serializer
68:         Serializer output = new Serializer(System.out);
69:         output.setIndent(2);
70:         output.write(doc);
71:     } catch (Exception exc) {
72:         System.out.println("Error: " + exc.getMessage());
73:         exc.printStackTrace();
74:     }
75: }
76: }

```

要测试该应用程序，可以使用一个来自报纸 *Toronto Star* 的 feed。下面的命令搜索标题中包含 Snow 的内容：

```
java RssFilter http://www.thestar.com/rss/000-082-672?searchMode=Lineup snow
```

应用程序源代码中的注释说明了代码的功能。

XOM 的设计遵循了一个首要原则：坚持尽可能简单。

在这个类库的网站上，Harold 指出：XOM 可帮助没有经验的开发人员做正确的事情，并避免它们做错误的事情。学习曲线不应陡峭，这包括不依赖于业界著名但并非显而易见的最佳实践。

对于需要在其程序中少量处理 XML 的 Java 程序员来说，这个新的类库很有帮助。

## 19.5 总结

在本章中，您探索了可扩展标记语言（XML）最流行的用途之一 RSS feed，从而学习了另一种流行的数据表示格式——XML 的基本知识。

在很多方面，XML 是数据领域的 Java 语言。它使数据独立于创建它的软件和该软件所在的操作系统，就像 Java 使软件独立于操作系统一样。

通过使用诸如开源的 XML 对象模型（XOM）等类库，可轻松地创建 XML 文件以及从中检索数据。

使用 XML 来表示数据的最大优点之一是，您总能将数据取回。如果您决定将数据移到关系型数据库、MySQL 数据库或其他数据源中，可以很容易地取回这些信息。在当前和未来，可使用软件以无数方式挖掘 RSS feed 中的数据。

您也可以使用各种技术（Java 或使用其他语句开发的工具）将 XML 转换为其他格式（如 HTML）。

## 19.6 问与答

问：RSS 1.0、RSS 2.0 和 Atom 之间有何不同？

答：RSS 1.0 是一种这样的联合格式，即使用资源描述框架 (RDF) 来描述 feed 中的内容。RSS 2.0 起源与 RSS 1.0 相同，但不使用 RDF。Atom 是另一种联合格式，它是在前面两种格式之后开发的，并被 IETF 采纳为 Internet 标准。

所有这 3 种格式都适合用于以 XML 格式提供 Web 内容，这种内容可被诸如 Bloglines 和 My Yahoo 等阅读器读取，也可被软件读取并存储、操作或变换。

问：为什么 Extensible Markup Language 被简称为 XML 而不是 EML？

答：该语言的创建者没有说明选择缩略语 XML 的原因。XML 团体中的普遍说法是，XML 比 EML “更酷”。在人们讥笑这种区别方式之前，Sun Microsystems 选择 Java 作为其编程语言的名称时使用了同样的标准，没有使用技术性更强的称谓，如 DNA 和 WRL。

也可能是这样的，XML 的创建者想避免与早期的编程语言 EML (Extended Machine Language) 混为一谈。

## 19.7 小测验

请回答下述问题，以复习本章介绍的内容。

### 19.7.1 问题

1. RSS 表示什么？
  - a. Really Simple Syndication,
  - b. RDF Site Summary,
  - c. 两者。
2. 使用 XOM 时，可通过哪个方法给 XML 元素添加文本？
  - a. addAttribute(String, String),
  - b. appendChild(Text),
  - c. appendChild(String)。
3. 在文档中以一致的方式使用所有的开始元素标记、结束元素标记和其他标记时，使用哪个形容词来描述该文档？
  - a. 验证的，
  - b. 可分析的，
  - c. 格式良好的。

答案

1. c, RSS 2.0 宣称它表示的是 Really Simple Syndication, 而 RSS 1.0 宣称它表示的是 RDF Site Summary。
2. b 和 c 都管用，一个以字符数据的方式添加 Text 元素的内容，另一个以字符串的方式添加。
3. c, 数据要成为 XML，必须是格式良好的。

### 19.7.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要

使用 Java 编译器对代码进行测试。

对于下述代码：

```
public class NameDirectory {
    String[] names;
    int nameCount;

    public NameDirectory() {
        names = new String[20];
        nameCount = 0;
    }

    public void addName(String newName) {
        if (nameCount < 20)
            // answer goes here
    }
}
```

NameDirectory 类必须能够存储 20 个不同的名称。要使这个类正确运行，应使用哪条语句替换 //answer goes here:

- a. names[nameCount] = newName;
- b. names[nameCount] == newName;
- c. names[nameCount++] = newName;
- d. names[++nameCount] = newName;

答案 c

## 19.8 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 创建一个简单的 XML 文档，用于表示包含 3 本图书的图书集。再创建一个 Java 应用程序，它搜索作者为 Joseph Heller 的图书并显示搜索结果。
2. 创建两个应用程序：一个检索数据库中的记录，并生成一个包含这些数据的 XML 文件；另一个读取该 XML 文件中的数据，并将其显示出来。





## 第 20 章

# XML Web 服务

多年来，人们无数次试图为远程过程调用（remote procedure calls, RPC）制定一种标准协议。RPC 是一台计算机程序通过网络（如 Internet）调用另一个程序中过程的一种方法。

通常，这些协议都是完全语言无关的，使得用诸如 C++ 等语言编写的客户程序可以调用用 Java 或其他语言编写的远程数据库服务器，而任何一方都无需知道也不关心对方的具体实现语言。

Web 服务以及使用 Web 以其他软件易于理解的格式提供数据的网络程序，正在以惊人的速度推动 RPC 的发展。Web 服务被用来在网站之间共享密码验证信息、在商店之间处理电子商务事务、实现 B2B（business-to-business）信息交换以及其他一些事情。

在这个领域中，最流行的技术之一是 XML-RPC，它是一种协议，使用超文本传输协议（HTTP）和可扩展标记语言（XML）进行远程过程调用。本章将介绍如何使用 Java 来实现这种技术，包括下列主题：

- XML-RPC 的发展历程；
- 如何使用 XML-RPC 与另一台计算机通信；
- 如何结构化 XML-RPC 请求和 XML-RPC 响应；
- 如何在 Java 程序中使用 XML-RPC；
- 如何发送 XML-RPC 请求；
- 如何接收 XML-RPC 响应。

### 20.1 XML-RPC 简介

Java 支持一种久负盛名的远程过程调用技术：远程方法调用（remote method invocation, RMI）。

RMI 与其他很多 RPC 技术（如 Common Object Request Broker Architecture（CORBA）和 SUN 公司的 Open Network Computing RPC）有一个相同的特点：复杂。它们都是为大量不同的远程计算任务设计的健壮解决方案。

这种完善性和复杂性阻碍了现有 RPC 技术的应用。实现这些解决方案的复杂程度远远超出了程序员所期望的内容：通过网络交换信息。

另一种更简单的替代方案是 XML-RPC，它已被 Web 服务广泛采用。

XML-RPC 很简单，迅速获得了 Red Hat、Microsoft 以及众多 UerTalk 和 Python 开发人员的支持。

大多数平台和被广泛使用的编程语言都支持 XML-RPC 的客户/服务器实现。UserLand Software 在其 XML-RPC.com 网站提供了一个实现目录，其网址为 <http://www.xmlrpc.com>。

XML-RPC 组合使用 HTTP 协议（万维网协议）和 XML（一种组织数据的格式，独立于用于读写数据的软件）来交换信息。

XML-RPC 支持下列数据类型。

- array：一个数据结构，存储多个其他数据类型（包括数据）的元素。

- base64: Base 64 格式的二进制数据。
- boolean: 布尔型值, 取值为 1 (true) 或 0 (false)。
- dateTime.iso8601: 一个包含日期和时间的字符串, 采用 ISO8601 格式 (如 20070915T19:20:15 表示 2007 年 9 月 15 日下午 7 点 20 分 15 秒)。

- double: 8 字节的带符号浮点数。
- int (也叫 i4): 带符号整数, 取值范围为 -2147483648 ~ 2147483647, 与 Java 语言中的 int 数据类型相同。

- string: 文本。
- struct: 由相关数据组成的名称-值组合, 其中名称为字符串, 值可以为任何其他数据类型 (类似于 Java 中的 HashMap 类)。

在 XML-RPC 中, 缺乏一种将数据存储为对象的方法。这种协议在最初设计时并没有考虑面向对象编程, 但使用 array 和 struct 类型可以表示相当复杂的对象。

XML-RPC 是一种非常适用于跨网络编程的远程过程调用协议。该协议已经成为 Windows、Macintosh、Linux 和 UNIX 系统上很多软件开发人员实现 Web 服务的关键元素之一。

#### 注意

完整的 XML-RPC 规范可在 <http://www.xml-rpc.com/spec> 找到。

有 75 种针对各种平台和语言的 XML-RPC 实现。

XML-RPC 发布后, 对该规范进行了扩展, 创建了另一种协议 SOAP (Simple Object Access Protocol, 简单对象访问协议)。

SOAP 与 XML-RPC 的有些设计目标相同, 但对 XML-RPC 进行了扩展, 以更好地支持对象、用户定义数据类型及其他高级功能, 从而产生了另一种复杂得多的协议。SOAP 协议在 Web 服务和其他分布式网络编程中也得到越来越多的应用。

#### 注意

既然 SOAP 是 XML-RPC 的扩展, 为何 XML-RPC 仍在使用的呢?

XML-RPC 规范发布后, 很多开发人员团体便迅速实现了它。SOAP 面世后, 人们认为它比 XML-RPC 复杂得多, 同时相关协议之间的差异也很大, 因此应根据项目的具体需求来选择其中的一种。

有关 SOAP 以及可与 SOAP 客户程序一起使用的公共服务器的更详细信息, 请访问网站 <http://www.xmethods.com>。

## 20.2 使用 XML-RPC 进行通信

XML-RPC 是一种通过 HTTP (用于在 Web 服务器与 Web 浏览器之间进行数据交换的标准) 进行传输的协议。它所传输的信息不是网页内容, 而是以特定方式编码的 XML。

XML-RPC 主要用于进行两种数据交换: 客户端请求和服务端响应。

### 20.2.1 发送请求

XML-RPC 请求是作为 HTTP POST 请求的一部分发送到 Web 服务器的 XML 数据。

POST 请求通常用于从 Web 浏览器传送数据到 Web 服务器: Java servlets、CGI 程序及其他从 POST 请求收集数据并发送 HTML 响应的软件。当从网页提交电子邮件或进行在线投票时, 可使用 POST 或另一种类似的 HTTP 请求——GET。

另一方面, XML-RPC 只是使用 HTTP 作为一种方便的协议与服务器通信并接收返回的响应。请求由两部分组成: POST 传输所需的 HTTP 报头及以 XML 格式表示的 XML-RPC 请求。程序清单 20.1 是一个 XML-RPC 请求示例。

程序清单 20.1 一个 XML-RPC 请求

```

1: POST /XMLRPC HTTP/1.0
2: Host: www.advogato.org
3: Connection: Close
4: Content-Type: text/xml
5: Content-Length: 151
6: User-Agent: OSE/XML-RPC
7:
8: <?xml version="1.0"?>
9: <methodCall>
10:   <methodName>test.square</methodName>
11:   <params>
12:     <param>
13:       <value>
14:         <int>13</int>
15:       </value>
16:     </param>
17:   </params>
18: </methodCall>

```

在程序清单 20.1 中, 第 1~6 行是 HTTP 报头, 第 8~18 行是 XML-RPC 请求。该程序清单包含如下信息:

- XML-RPC 服务器位于 <http://www.advogato.org/XMLRPC> (第 1~2 行);
- 被调用的远程方法是 test.square (第 10 行);
- 调用该方法时使用了一个参数——整数 13 (第 12~16 行)。

与 Java 不同, XML-RPC 请求中的方法名称不包括括号。它们由对象名称、句点和方法名称组成, 或只包含方法名称 (这取决于 XML-RPC 服务器)。

#### 警告

在无数种计算机程序语言中得到实现了的 XML-RPC 协议, 在术语方面与 Java 有几个不同之处: 方法 (method) 被称为过程 (procedures), 而方法参数 (method arguments) 则被称为参数 (parameters)。本章介绍 Java 编程技术时将主要使用 Java 术语。

## 20.2.2 响应请求

XML-RPC 响应是从 Web 服务器返回的 XML 数据, 像任何其他 HTTP 响应一样。同样, XML-RPC 在已建立的进程上返回响应——正如 Web 服务器借助于 HTTP 发送数据给 Web 浏览器, 并以一种新的方式使用它。

响应也由 HTTP 报头和 XML 格式的 XML-RPC 响应组成。

程序清单 20.2 是一个 XML-RPC 响应示例。

程序清单 20.2 XML-RPC 响应

```

1: HTTP/1.0 200 OK
2: Date: Sun, 03 Mar 2007 19:17:11 GMT
3: Server: Apache/1.3.26 (Unix) mod_virgule/1.41 PHP/4.1.2 mod_perl/1.26
4: ETag: "PbT9cMgXsXnw520qREFNAA=="
5: Content-MD5: PbT9cMgXsXnw520qREFNAA==
6: Content-Length: 157
7: Connection: close
8: Content-Type: text/xml
9:
10: <?xml version="1.0"?>

```

```

11: <methodResponse>
12:   <params>
13:     <param>
14:       <value>
15:         <int>169</int>
16:       </value>
17:     </param>
18:   </params>
19: </methodResponse>

```

在程序清单 20.2 中，第 1~8 行是 HTTP 报头，第 10~19 行是 XML-RPC 响应。从该程序清单中可了解到下列信息：

- 响应长 157 字节，为 XML 格式（第 6 行和第 8 行）。
- 远程方法返回的值为整数 169（第 15 行）。

XML-RPC 响应只包含一个参数，这与第 12 行中的 `params` 标记不太相符。如果远程方法没有返回值（如它可能是一个返回 `void` 的 Java 方法），XML-RPC 服务器仍将返回一些信息。

返回值可以是基本数据类型、字符串、一维或多维数组及其他复杂数据结构（如键-值组合，在 Java 中用 `HashMap` 或其子类来实现）。

#### 注意

上述 XML-RPC 请求和响应示例都是由开源网站 Advogato 运行的一台服务器生成的。关于其 XML-RPC 服务器的其他信息，请访问 <http://www.advogato.org/xmlrpc.html>。

网上有多种 XML-RPC 调试器，可用于调用远程方法并查看完整的 XML-RPC 请求和响应，这使得判断客户端或服务器是否正确运行容易得多。

## 20.3 选择 XML-RPC 实现

虽然可创建自己的类来读写 XML 以及在 Internet 上交换数据，以利用 XML-RPC 技术，但更好的方法是使用支持 XML-RPC 的 Java 类库。

其中最流行的一个是 Apache XML-RPC，它是由 Apache Web 服务器、Tomcat Java servlet 引擎、Ant 构建工具和其他开源软件的开发人员管理的一个开源项目。

Apache XML-RPC 由 `org.apache.xmlrpc` 包和 3 个相关的包组成。通过其中的类，您只需编写很少的代码便可实现 XML-RPC 客户端和服务端。

该项目有一个主页，网址为 <http://xml.apache.org/xmlrpc>，当前的发布版为 2.0。

要使用该项目，必须下载并安装一个存档文件，其中包括两个 JAR 文件：`xmlrpc-2.0.jar` 和 `xmlrpc-1.2b1-applet.jar`。

这个安装存档文件可作为 zip 文件下载（对于 Windows 用户）或作为组合的 TAR/GZ 格式文件下载（对于 Linux、UNIX 和 Mac OS X 用户）。

根据您的操作系统，下载并解压相应的存档文件。主文件夹中包含两个 JAR 存档文件：`xmlrpc-2.0.jar` 和 `xmlrpc-2.0-applet.jar`（您安装时版本号可能不同），这些文件包含 Apache XML-RPC 类库。

解压缩文件后，在系统的环境变量 `Classpath` 中添加对这两个 JAR 文件的引用，让 Java 解释器和编译器能够找到 Apache XML-RPC 的包。

必须将这两个文件的完整文件夹位置和名称添加到环境变量 `Classpath` 中的某个地方。例如，在 Windows 平台上，如果文件的位置为 `C:\jdk\xmlrpc\xmlrpc-2.0.jar` 和 `C:\jdk\xmlrpc\xmlrpc-2.0-applet.jar`，则应将下列文本添加到环境变量 `Classpath` 的最后：

```
;C:\jdk\xmlrpc\xmlrpc-2.0.jar;C:\jdk\xmlrpc\xmlrpc-2.0-applet.jar
```

其中分号用于分隔 Classpath 中的文件或文件夹。在 Linux 或 UNIX 系统上类似，但是文件或文件夹之间的分隔符为冒号（而不是分号）：

```
:C:\jdk\xmlrpc\xmlrpc-2.0.jar:C:\jdk\xmlrpc\xmlrpc-2.0-applet.jar
```

注意不要删除环境变量 Classpath 中的已有路径。关于如何设置该环境变量的更详细信息，请参见附录 A。

配置 Classpath 后，即可在 Java 程序中使用 Apache XML-RPC 类。为引用这些类，最容易的方法是使用 import 语句，使整个包可用，如下列语句所示：

```
import org.apache.xmlrpc.*;
```

它使得可以直接引用主包 org.apache.xmlrpc 中的类，而不需要使用完整的包名称。接下来的两节将介绍如何使用这个包。

#### 注意

如果 Apache XML-RPC 不符合您的要求，还有 20 多种其他实现可供选择。XML-PRC.com 提供用 Java、C++、PHP 和其他语言实现的 XML-RPC 实现清单。

## 20.4 使用 XML-RPC Web 服务

XML-RPC 客户端是一个程序，它连接到服务器，调用服务器的程序的方法，并存储返回的结果。

使用 Apache XML-RPC 时，其过程类似于调用 Java 中的任何其他方法：不需要创建 XML 请求、分析 XML 响应或使用 Java 的网络类连接到服务器。

在 org.apache.xmlrpc 包中，XmlRpcClient 类代表一个客户端。可用 3 种方式创建 XmlRpcClient 对象，每种方式都需要指定服务器的 URL：

- XmlRpcClient(String)：创建一个客户端，连接到 String 指定的地址，它必须是一个有效的网址（如 http://www.example.com）或有效的网址和端口号（如 http://www.example.com:2274）。
- XmlRpcClient(URL)：创建一个客户端，连接到指定的 URL 对象。
- XmlRpcClient(String, int)：创建一个客户端，连接到指定主机名 (String) 和端口号 (int)。

需要 String 参数的两个构造函数在该参数不是有效的 URL 时将引发异常 java.net.MalformedURLException。

下列语句创建一个客户端，连接到主机 cadenhead.org 的端口 4413 上的 XML-RPC 服务器：

```
XmlRpcClient client = new XmlRpcClient("http://cadenhead.org:4413");
```

如果使用任意数目的参数调用远程方法，应将参数存储在 Vector 对象（一种用于保存类对象的数据结构）中。

#### 注意

矢量在第 8 章中介绍过，它们是 java.util 包的一部分。

要使用矢量，可不带参数调用 Vector() 方法，然后调用它的 addElement(Object) 方法，将每个对象添加到该 Vector 中。对象可以是任何类，且必须按远程方法调用的次序添加它们。

下列数据类型可作为远程方法的参数：

- byte[] 数组，用于存储 base64 数据；
- Boolean 对象，用于存储布尔值；
- Date 对象，用于存储 dateTime.iso8601 值；
- Double 对象，用于存储 double 值；

- Integer 对象，用于存储 int 值；
- String 对象，用于存储 string；
- Hashtable 对象，用于存储 struct；
- Vector 对象，用于存储数组。

其中 Data、Hashtable 和 Vector 类都在 java.util 包中。

例如，如果 XML-RPC 服务器有一个方法，接受一个 String 和一个 double 参数，下列代码将创建一个 Vector，并保存这些参数：

```
String code = "conical";
Double xValue = new Double(175);
Vector parameters = new Vector();
parameters.addElement(code);
parameters.addElement(xValue);
```

要调用 XML-RPC 服务器上的远程方法，可使用下列两个参数调用 XmlRpcClient 对象的 execute(String, Vector) 方法：

- 方法的名称；
- 存储方法参数的矢量。

指定方法名称时不要用任何括号或参数。XML-RPC 服务器通常公开其包含的方法，例如，在作者的 Internet 服务器 4413 端口上运行了一个名为 cadenhead.org XML-RPC 服务器（这是作者的测试服务器）。它提供了 dmoz.getRandomSite() 方法，该方法返回一个 Object，其中包含关于一个随机网站的信息。该方法没有参数。

下列语句创建一个 XML-RPC 客户端，并调用该方法：

```
XmlRpcClient client = new XmlRpcClient("http://cadenhead.org:4413");
Vector params = new Vector();
Object result = client.execute("dmz.getRandomSite", params);
```

方法 execute() 返回一个包含响应信息的 Object。应将该对象转换为下列数据类型之一，然后将其作为参数发送给方法：Boolean、byte[]、Date、Double、Integer、String、Hashtable 或 Vector。

与 Java 中的其他网络方法一样，如果在客户端与服务器端连接期间发生输入/输出错误，将引发 java.net.IOException 异常。当服务器报告 XML-RPC 错误时，也将引发 XmlRpcException 异常。

方法 execute() 返回的对象的数据类型如下：对于布尔型 XML-RPC 值，为 Boolean；对于 base64 数据，为 byte[]；对于 dateTime.iso8601，为 Date；对于双精度数据，为 Double；对于 int (i4) 值，为 Integer；对于字符串，为 String；对于 struct 值，为 Hashtable；对于数组，为 Vector。

为在一个实际可运行的程序中查看这些内容，将程序清单 20.3 中的文本输入到文本编辑器中，并将其保存为 SiteClient.java。

程序清单 20.3 SiteClient.java 的完整源代码

```
1: import java.io.*;
2: import java.util.*;
3: import java.net.*;
4: import org.apache.xmlrpc.*;
5:
6: public class SiteClient {
7:     String url;
8:     String title;
9:     String description;
10:
11:     public static void main(String arguments[]) {
12:         SiteClient client = new SiteClient();
13:         try {
14:             Vector response = client.getRandomSite();
15:             // Report the results
16:             if (response.size() > 0) {
```

```

17:         client.url = response.get(1).toString();
18:         client.title = response.get(2).toString();
19:         client.description = response.get(3).toString();
20:         System.out.println("URL: " + client.url
21:             + "\nTitle: " + client.title
22:             + "\nDescription: " + client.description);
23:     }
24: } catch (IOException ioe) {
25:     System.out.println("IO Exception: " + ioe.getMessage());
26:     ioe.printStackTrace();
27: } catch (XmlRpcException xre) {
28:     System.out.println("XML-RPC Exception: " + xre.getMessage());
29: }
30: }
31:
32: public Vector getRandomSite()
33:     throws IOException, XmlRpcException {
34:
35:     // Create the client
36:     XmlRpcClient client = new XmlRpcClient(
37:         "http://localhost:4413/");
38:     // Create the parameters for the request
39:     Vector params = new Vector();
40:     // Send the request and get the response
41:     Vector result = (Vector) client.execute("dmoz.getRandomSite",
42:         params);
43:     return result;
44: }
45: }

```

应用程序 SiteClient 建立一个到 XML-RPC 服务器的连接，并不带参数调用服务器上的 dmoz.getRandomSite() 方法。该方法返回一个 Vector 对象，其中包含 4 个字符串：ok（表明请求已实现）以及网站的 URL、标题和描述。

SiteClient 应用程序的输出类似如下：

```

URL: http://www.rdb.com/
Title: rdb
Description: Makers of a very simple quasi-relational database based on Unix
shell commands

```

## 20.5 创建 XML-RPC Web 服务

XML-RPC 服务器是一个程序，接收来自客户端的请求，通过调用来响应客户端请求，然后返回结果。服务器维护一个允许客户端调用的方法列表，它们是 Java 类，被称为处理程序。

Apache XML-RPC 负责处理所有 XML 和网络，使得您可将注意力全部集中在希望远程方法完成的任务上。

使用 org.apache.xmlrpc 包中的类提供远程方法服务的方式有几种。最简单的方式是使用 WebServer 类，它表示一个简单的 HTTP Web 服务器，只响应 XML-RPC 请求。

这个类有两个构造函数：

- WebServer(int)：创建一个 Web 服务器，监听指定端口。
- WebServer(int, InetAddress)：在指定端口和 IP 地址创建一个 Web 服务器。第 2 个参数是一个 java.net.InetAddress 类对象。

如果在创建和启动服务器时产生输入/输出错误，则这两个构造函数都将引发 IOException 异常。

下列语句在端口 4413 上创建一个 XML-RPC Web 服务器：

```

int port = Integer.parseInt("4413");
WebServer server = new WebServer(port);

```

该 Web 服务器不包含客户端通过 XML-RPC 调用的远程方法。这些方法位于其他的 Java 类中，它们被称为处理程序。

添加处理程序的方法是调用服务器的 `addHandler(String, Object)` 方法，并在参数中指定处理程序名称和处理程序对象。

方法 `addHandler()` 的第一个参数是处理程序的名称，它可以为任何内容：与变量名称类似。客户端在调用远程方法时将使用该名称。

本章前面创建的 `SiteClient` 应用程序调用了远程方法 `dmoz.getRandomSite()`。该调用的第一部分（句点前面的文本）引用了一个名为 `dmoz` 的处理程序。

方法 `addHandler()` 的第 2 个参数是一个类对象，其公用方法可供远程调用。

下列语句为一个名为 `WebServer` 的对象创建一个处理程序：

```
DmozHandler odp = new DmozHandler();
server.addHandler( "dmoz", odp);
```

该示例中的处理程序是一个 `DmozHandler` 对象，它包含一个 `getRandomSite()` 方法，该方法返回 Open Directory Project 数据库中一个随机网站的相关信息。后面将创建这个类。

用于处理远程方法调用的类可以是任何包含 `public` 方法（这种方法有返回值）、接受 Apache XML-RPC 支持的数据类型（`boolean`、`byte[]`、`Date`、`double`、`Hashtable`、`int`、`String` 和 `Vector`）作为其参数的 Java 类。

可以很容易地将已有 Java 类用作 XML-RPC 处理程序而不需对它进行修改，只要它们不包含不会被调用的 `public` 方法且每个 `public` 方法返回合适的值。

#### 警告

返回值是否合适取决于 Apache XML-RPC 实现（而不是 XML-RPC 本身）。该协议的其他实现很可能在远程方法调用中所接受的参数数据类型和返回值数据类型方面有所不同。

通过使用 Apache XML-RPC，Web 服务器将允许处理程序中的任何 `public` 方法被调用，因此应该使用访问控制来限制调用远程方法的客户端。

作为创建 XML-RPC 服务的第一步，下列代码创建了一个简单的 Web 服务器，它可接收 XML-RPC 请求。输入程序清单 20.4 中的文本，并将其保存为 `DmozServer.java`。

#### 程序清单 20.4 DmozServer.java 的完整源代码

```
1: import java.io.IOException;
2: import org.apache.xmlrpc.WebServer;
3: import org.apache.xmlrpc.XmlRpc;
4:
5: public class DmozServer {
6:     public static void main(String[] arguments) {
7:         if (arguments.length < 1) {
8:             System.out.println("Usage: java DmozServer [port]");
9:             System.exit(0);
10:        }
11:        try {
12:            startServer(arguments[0]);
13:        } catch (IOException ioe) {
14:            System.out.println("Server error: " +
15:                ioe.getMessage());
16:        }
17:    }
18:
19:    public static void startServer(String portString) throws IOException {
20:        // Start the server
21:        int port = Integer.parseInt(portString);
22:        System.out.println("Starting Dmoz server ...");
```





```

23:         WebServer server = new WebServer(port);
24:
25:         // Register the handler
26:         DmozHandler odp = new DmozHandler();
27:         server.addHandler("dmoz", odp);
28:         server.start();
29:         System.out.println("Accepting requests ...");
30:     }
31: }

```

在创建处理程序类 `DmozHandler` 之前，上述类将无法通过编译。

应用程序 `DmozServer` 接受一个端口号作为命令行参数，并用该参数调用 `startServer()` 方法。

`startServer()` 方法创建一个 `WebServer` 对象，在指定端口号上监视进入的 XML-RPC 请求。一个处理程序被添加到该服务器：名称为 `dmoz` 的 `DmozHandler` 对象，然后调用服务器的 `start()` 方法，以开始监听请求。

这是实现 XML-RPC 服务器所需要的全部代码。多数工作都位于客户端要调用的远程方法中，不需要任何特殊技巧，只要它们都是 `public` 的，且返回一个适当的值。

为了提供一个完整的示例，便于您对它进行修改以满足自己的需求，下面提供了 `DmozHandler` 类。其中采用的所有技术都在第 18 章中介绍过，同时通过它可以很好地复习如何使用 JDBC 从数据库中检索记录：这个类使用了一个名为 `db1` 的 MySQL 数据库。

输入程序清单 20.5 中的文本，并将其保存为 `DmozHandler.java`，然后编译类 `DmozServer.java` 和 `DmozHandler.java`。

程序清单 20.5 `DmozHandler.java` 的完整源代码

```

1: import java.sql.*;
2: import java.util.*;
3:
4: public class DmozHandler {
5:     public Vector getRandomSite() {
6:         Connection conn = getMySQLConnection();
7:         Vector<String> response = new Vector<String>();
8:         try {
9:             Statement st = conn.createStatement();
10:            ResultSet rec = st.executeQuery(
11:                "SELECT * FROM cooldata ORDER BY RAND() LIMIT 1");
12:            if (rec.next()) {
13:                response.addElement("ok");
14:                response.addElement(rec.getString("url"));
15:                response.addElement(rec.getString("title"));
16:                response.addElement(rec.getString("description"));
17:            } else {
18:                response.addElement("database error: no records found");
19:            }
20:        } catch (SQLException sqe) {
21:            response.addElement("database error: " + sqe.getMessage());
22:        }
23:        return response;
24:    }
25:
26:    private Connection getMySQLConnection() {
27:        Connection conn = null;
28:        String data = "jdbc:mysql://localhost/cool";
29:        try {
30:            Class.forName("com.mysql.jdbc.Driver");
31:            conn = DriverManager.getConnection(
32:                data, "username", "password");
33:        } catch (SQLException s) {
34:            System.out.println("SQL Error: " + s.toString() + " "
35:                + s.getErrorCode() + " " + s.getSQLState());
36:        } catch (Exception e) {
37:            System.out.println("Error: " + e.toString()
38:                + e.getMessage());

```

```

39:         }
40:         return conn;
41:     }
42: }

```

应对应用程序 DmozHandler 的第 28~32 行进行修改, 替换成您自己的 JDBC 或 JDBC-ODBC 驱动程序、用户名和密码。另外, 还需要根据使用的驱动程序, 修改该字符串的其他部分, 以便能够连接到这个数据库。

通过在命令行参数中指定端口号, 来运行该服务器程序, 如下所示:

```
java DmozServer 4413
```

当服务器程序启动并运行后, 即可修改应用程序 SiteClient 以连接到您自己的 XML-RPC 服务器。修改程序清单 20.3 中的第 36~37 行, 用实际的服务器引用、冒号和端口号替换 localhost:4413, 如下所示:

```

XmlRpcClient client = new XmlRpcClient(
    "http://cadenhead.org:4413/");

```

## 20.6 总结

XML-RPC 曾被认为是远程过程调用协议的“最低标准”, 但它的创始人并不认为这是一种贬低。很多用于帮助软件通过网络进行通信的技术太复杂, 吓退了很多只有简单需求的开发人员。

XML-RPC 协议可用于与任何支持 HTTP (Web 通用语言) 和 XML (一种更高级的人类可读的数据格式) 的软件交换信息。

通过查看 XML-RPC 请求和响应, 可了解到如何使用该协议, 甚至无需阅读该协议规范。

然而, 由于有些实现 (如 Apache XML-RPC) 已经应用相当广泛, 使得即使不了解该协议也可迅速地掌握如何使用它。

## 20.7 问与答

问: 当我试图从远程方法返回一个 HashMap 对象时, Apache XML-RPC 返回 XmlRpcException 异常, 指出不支持这种对象。它支持哪些对象?

答: Apache XML-RPC 返回下列数据类型: Boolean (布尔型 XML-RPC 值)、byte[] (base64 数据)、Data (dateTime.iso8601 数据)、Double (双精度值)、Integer (int 或 i4 值)、String (字符串)、Hashtable (结构)、Vector (数组)。

问: 我正在编写一个 XML-RPC 客户端, 它调用一个返回二进制数据 (base64) 的方法。XmlRpcClient 的 execute() 方法返回一个对象 (而不是 byte 数组)。如何进行这种转换?

答: 数组在 Java 中是对象, 因此可通过强制转换将 execute() 方法返回的对象转换为 byte 数组 (假定该对象实际上是一个数组)。下面的语句对一个包含 byte 数组、名为 fromServer 的对象进行这种转换:

```
byte[] data = (byte[]) fromServer;
```

## 20.8 小测验

请回答下述 3 个问题，以复习本章介绍的内容。

### 20.8.1 问题

1. 下列哪一种流行的 Internet 协议不是 XML-RPC 需要的协议？
  - a. HTML;
  - b. HTTP;
  - c. XML。
2. 哪种 XML-RPC 数据类型最适合用于存储数字 8.67？
  - a. boolean;
  - b. double;
  - c. int。
3. 哪个 XML 标记指出数据是 XML-RPC 请求？
  - a. methodCall;
  - b. methodResponse;
  - c. params。

#### 答案

1. a, XML-RPC 使用 HTTP 来传输 XML 格式的数据。它不使用 HTML。
2. b, 在 XML-RPC 中，所有浮点数（如 8.67）都用 double 类型表示。它不像 Java 那样有两种不同的浮点数类型（float 和 double）。
3. a, methodCall 标记只用于请求中，methodResponse 只用于响应中，而 params 则可用于两者中。

### 20.8.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题，请回答该问题，而不要查看本章的内容，也不要使用 Java 编译器对代码进行测试。

给定如下代码：

```
public class Operation {  
    public static void main(String[] arguments) {  
        int x = 1;  
        int y = 3;  
        if ((x != 1) && (y++ == 3))  
            y = y + 2;  
    }  
}
```

y 的最终值是多少？

- a. 3;
- b. 4;
- c. 5;
- d. 6。

答案 a



## 20.9 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 编程网站 Advogato 提供了一个 XML-PRC 接口，可用于读取成员的日记，其网址为 <http://www.advogato.org/xmlrpc.html>。请编写一个应用程序，它读取一位成员的最后 10 篇日记。
2. Weblogs.com 在 <http://www.weblogs.com/api.html> 提供了一个 XML-PRC 接口，该接口提供了博客更新服务。请编写一个客户端和服务端，它们能够发送和接收 weblogUpdates.ping 方法。



## 第 21 章

# 编写 Java servlet 和 Java Server Page

本书探讨的最后一个主题是 Java 中最为引人入胜且在不断发展变化的领域之一：将 Web 服务器用作应用程序开发平台。

Java 不仅被用于编写运行在计算机上的应用程序以及运行在网页中的小程序。servlet 是服务器通过 Internet 运行的 Web 应用程序，并通过 Web 浏览器呈现给用户，它使用了 Java 的所有功能。

通过使用 servlet，可以通过网页表单收集用户的输入，将数据库或其他数据源中的数据呈现给用户，以及动态地创建网页。

通过使用 JavaServer Page (JSP)，可以进一步改善这种方法。JSP 是一种结合使用静态 HTML 及 servlet 和 Java 表达式的输出来创建网页的方式。

JSP 让非程序员能够维护使用 Java 开发的网站。

本章介绍以下主题：

- servlet 不同于应用程序和小程序的地方；
- 如何将 servlet 作为 Apache Web 服务器和其他服务器的一部分来运行；
- 如何接收 Web 表单中的数据；
- 如果存储和检索 cookie；
- 如何使用 servlet 动态地生成网页内容；
- 如何使用 JSP 开发 Web 应用程序；
- 如何在网页中使用 Java 变量、表达式和语句。

### 21.1 使用 Web servlet

servlet 是由这样的 Web 服务器运行的 Java 类，即安装有支持 Java servlet 规范的解释器。该解释器通常被称为 servlet 引擎，被优化成使用最少的 Web 服务器资源来运行 servlet。

Java servlet 的用途与使用通用网关接口 (Common Gateway Interface, CGI) 实现的程序相同。CGI 是一种协议，用于编写通过 Web 服务器来发送和接收信息的软件。几乎从万维网面世起，它便支持 CGI 编程。大多数 CGI 程序 (CGI 脚本) 都是使用诸如 Perl、Python 和 PHP 等语言编写的。

CGI 程序的用途如下：

- 收集 Web 表单中的用户输入；
- 接收统一资源定位符 (URL) 中以参数方式指定的信息；
- 在运行 Web 服务器的计算机上运行程序；
- 存储和检索 cookies——Web 用户计算机上用于存储其偏好和其他信息的文件；
- 以 HTML 文档、GIF 图形或其他格式将数据发回给 Web 用户。

Java servlet 能够完成上述所有任务，同时支持一些使用大多数 CGI 脚本语言都难以实现的行为。

servlet 对会话提供了全面支持,会话是一种跟踪 Web 用户在特定时段如何浏览网站的方式。servlet 还能使用标准接口直接与 Web 服务器进行通信。只要服务器支持 Java servlet,它就能与这些程序交换信息。

与 Java 本身一样,Java servlet 也有可移植性的优点。虽然 Sun 的 servlet 实现是同 Apache 软件基金会(一家开源开发商,创建了 Apache Web 服务器)一道创建的,但其他服务器开发商也提供了支持 Java servlet 的工具,这包括 IBM WebSphere、BEA WebLogic 和 Jetty Server。

另外,servlet 在内存方面也是高效的。如果 10 个人同时使用同一个 CGI 脚本,Web 服务器将把该脚本的 10 个拷贝加载到内存中。然而,如果 10 个人同时使用一个 Java servlet,则只有该 servlet 的一个拷贝被加载,该 servlet 将生成线程来处理每个用户。

要创建 servlet,可使用 javax.servlet 和 javax.servlet.http 包,它们是 Java 企业版的标准组成部分。Java 企业版是 Java 类库的扩展版本,支持大型项目开发。

这些类实现了 Java Servlet 和 JSP 规范。当前,最新的版本是 Java Servlet 2.4 和 JSP 2.1。

对于使用 Java 标准版(本书介绍的内容针对的都是这个版本)的用户,可从 Sun 的 Java servlet 网站(<http://java.sun.com/products/servlet>)下载 servlet 包。单击链接 Downloads,然后单击标题 Specifications 下针对 Java Servlet 规范 2.3 版的下载链接(也可以下载更新版本的类文件)。

安装类文件后,为使它们可用,将其安装目录加入到环境变量 Classpath 中。例如,如果文件夹 javax 位于 C:\java\servlet-2.3 中,则将文件夹 C:\java\servlet-2.3 加入到 Classpath 中。

另一种获取这些文件的方式是,使用将用于部署应用程序的服务器自带的 Java servlet 类库。

多种 Web 服务器都支持 servlet,它们的安装、安全性和管理步骤各不相同。

对于新的 servlet 开发人员来说,最流行的选择是 Tomcat,它是 Apache 软件基金会和 Sun Microsystems 合作开发的一种开源服务器。Tomcat 5.5 版支持 Java Servlet 2.4 和 JSP 2.0。

Tomcat 可与其他 Web 服务器(如 Apache 的 Web 服务器)一起运行,也可作为独立的服务器运行。读者的 Web 服务器或 Web 应用程序服务器可能支持这些 servlet。

Tomcat 可免费从 Apache 的网站下载,其网址为 <http://jakarta.apache.org/tomcat>。可供下载的版本有多个,请下载编号为 5.5 版。

该网站还提供了完整的 Tomcat 安装说明。如果要将其作为独立的服务器运行,并用于测试目的,则在大部分情况下,可按下述步骤来安装它:

1. 进入针对 Tomcat 和其他 Apache 项目的下载页面。
2. 在网络上下载 Tomcat 5.5。该软件可以 ZIP 存档文件、使用 TAR 和 GZ 压缩的存档文件或 Windows 安装程序的方式下载。
3. 安装该软件,并记下其安装目录。
4. 创建一个名为 JAVA\_HOME 的环境变量,它包含 Java 的安装目录。
5. 创建一个名为 CATALINA\_HOME 的环境变量,它包含 Tomcat 的安装目录。
6. 在 Tomcat 安装目录中的子目录 bin 中,使用 catalina.sh 或 catalina.bat 来运行该服务器,相应的命令如下:  

```
catalina.sh start
catalina.bat start
```

 Tomcat 将运行在端口 8080 上。要关闭它,可执行命令 stop。
7. 要验证它是否在运行,可使用 Web 浏览器访问 <http://localhost:8080>。如果从另一台计算机上测试 Tomcat,应将 localhost 替换为安装 Tomcat 的服务器的域名或 IP 地址。
8. 在环境变量 Classpath 中,添加到 Tomcat 自带的 Java servlet 类库 servlet-api.jar 的引用。在 5.5 版中,该类库位于子文件夹 common\lib 中。

如果您没有服务器，又想开发 servlet，有几个公司提供支持 Java servlet 的商业 Web 托管服务。这些公司安装了 Tomcat，且对其进行了配置，使之能在服务器上正常工作，您只需使用 javax.servlet 和 javax.servlet.http 包中的类来编写 servlet 即可。

## 21.2 开发 servlet

Java servlet 的创建和编译方式与其他 Java 应用程序相同。安装上述两个 servlet 包，并将其所在的文件夹加入到 CLASSPATH 中后，便可以使用 JDK 的 Java 编译器或其他编译器来编译它们。

所有 servlet 都是 HttpServlet 类的子类，后者位于 javax.servlet 包中。这个类包含用于表示 servlet 的生命周期以及从运行 servlet 的 Web 服务器那里接收信息的方法。

Servlet 的生命周期方法的功能类似于小程序的生命周期方法。

当 Web 服务器运行 servlet 以处理用户请求时，方法 `init(servletConfig)` 将自动被调用。正如前面指出的，同一个 Java servlet 能够处理来自不同 Web 用户的多个请求。`init()` 方法只被调用一次，即 servlet 启动时。如果请求时 servlet 正在运行，则不会再次调用 `init()` 方法。

方法 `init()` 接受一个参数：`servletConfig`——javax.servlet 包中的一个接口，其中包含用于了解关于 servlet 运行环境的详细信息的方法。

Web 服务器终止 servlet 时，方法 `destroy()` 会被调用。像方法 `init()` 一样，该方法也只被调用一次——所有用户都接收完毕来自 servlet 的信息时。如果指定的时间内没有接收完毕，`destroy()` 将自动被调用，以防止 servlet 因等待与用户交换信息而被挂起。

servlet 的主要任务之一是收集来自 Web 用户的信息，并提供相应的内容进行响应。您可以使用表单来收集来自用户的信息，表单是网页上的一组文本框、单选按钮、文本区域、按钮和其他输入框。

图 21.1 显示了页面上的一个 Web 表单，该页面是使用 Web 浏览器 Mozilla 加载的。

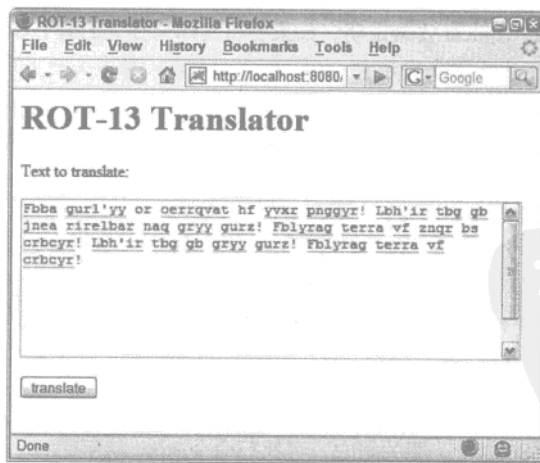


图 21.1 使用 Web 表单收集信息

图 21.1 中的表单包含两个控件：一个文本区域和一个名为 Translate 的可单击按钮。用于显示该页面的 HTML 标记如下：

```

<html>
<body>
<head><title>ROT-13 Translator</title></head>
<h1>ROT-13 Translator</h1>
<p>Text to translate:
<form action="Rot13" method="post">
<textarea name="text" rows="8" cols="55">
</textarea>
<p><input type="submit" value="translate">
</form>
</body>
</html>

```

表单包含在 HTML 标记<form>和</form>的之间。表单中的控件由其自己的标记表示：文本区域为<textarea>和</textarea>；Translate 按钮为<input>。文本区域的名称为 text。

#### 提示

要开发 Web servlet，必须对 HTML 有基本的了解，因为 servlet 唯一的用户界面是运行在浏览器中的网页。下面是本非常适合用于学习 HTML 的图书：《Html 与 CSS 入门经典》。

表单中的每个控件中都存储了信息，这些信息可被传递给 Web 服务器，进而传递给 Java servlet。Web 浏览器使用 HTTP 与服务器进行通信。可以使用两种 HTTP 请求来将表单数据发送给服务器：GET 和 POST。

网页使用 GET 或 POST 调用服务器时，必须将处理请求的程序的名称指定为 Web 地址——统一资源定位符（URL）。

GET 请求将表单中的所有数据追加到 URL 的后面，如下例所示：

```
http://www.java21days.com/servlets/beep?number=5551220&repeat=no
```

POST 请求将表单数据作为一个报头，该报头将独立于 URL 被发送。这通常更好，而如果从表单中收集的是机密信息，则必须这样做。另外，有些 Web 服务器和浏览器不支持超过 255 个字符的 URL，这限制了以 GET 请求发送的信息量。

对于这两种请求，Java servlet 都通过从 HttpServlet 类继承来的方法进行处理：*doGet(HttpServletRequest, HttpServletResponse)*和*doPost(HttpServletRequest, HttpServletResponse)*。这些方法可能引发两种异常：*servletException* 和 *IOException*，前者位于 *javax.servlet* 包中，而后者位于涉及输入/输出流的标准 *java.io* 包中。

方法 *doGet()* 和 *doPost()* 接受两个参数：一个 *HttpServletRequest* 对象和一个 *HttpServletResponse* 对象。前者在使用 GET 请求来执行 servlet 时被调用，后者在使用 POST 请求来执行 servlet 时被调用。在 Java servlet 编程中常用的一种技巧是，使用一个方法来调用另一个，如下所示：

```

public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    doPost(request, response);
}

```

请求对象和响应对象所属的类位于 *javax.servlet.http* 包中。servlet 通过调用类 *HttpServletRequest* 的方法接收有关它如何运行的信息。例如，当 Web 表单被提交给 servlet 时，表单中的每个控件的内容都被 *HttpServletRequest* 类存储为一个字符串。

servlet 通过发回 HTML 文档、图像文件或浏览器支持的其他类型的信息来与用户进行通信。它通过调用 *HttpServletResponse* 类的方法来发送这些信息。

准备响应时，首先要定义 servlet 将把什么类型的内容发送给浏览器。为此，可以调用方法 *setContentType(String)*，并将内容类型作为参数传递给它。

最常见的响应格式是 HTML，这可以通过调用 *setContentType("text/html")* 来设置。您还可以将响应作为文本（“text/plain”）、图形文件（“image/gif”、“image/jpeg”）和应用程序特定的格式（如



“application/msword”) 来发送。

要将数据发送给浏览器, 需要创建一个与浏览器相关联的 servlet 输出流, 然后调用该输出流的 `println(String)` 方法。servlet 输出流由 `ServletOutputStream` 类表示, 这个类位于 `javax.servlet` 包中。可以通过调用 `HttpServletResponse` 类的 `getOutputStream()` 方法来获得这样的流。

下面的例子使用 `HttpServletResponse` 对象 `response` 创建一个 servlet 输出流, 然后将一个简短的网页发送到这个流中:

```
ServletOutputStream out = response.getOutputStream();
out.println("<html>");
out.println("<body>");
out.println("<h1>Hello World!</h1>");
out.println("</body>");
out.println("</html>");
```

程序清单 21.1 中的 Java servlet 接收来自图 21.1 所示的表单中的数据。

程序清单 21.1 完整的 Rot13.java 源代码

```
1: import java.io.*;
2:
3: import javax.servlet.*;
4: import javax.servlet.http.*;
5:
6: public class Rot13 extends HttpServlet {
7:
8:     public void doPost(HttpServletRequest req, HttpServletResponse res)
9:         throws ServletException, IOException {
10:
11:         String text = req.getParameter("text");
12:         String translation = translate(text);
13:         res.setContentType("text/html");
14:         ServletOutputStream out = res.getOutputStream();
15:         out.println("<html>");
16:         out.println("<body>");
17:         out.println("<head><title>ROT-13 Translator</title></head>");
18:         out.println("<h1>ROT-13 Translator</h1>");
19:         out.println("<p>Text to translate:</p>");
20:         out.println("<form action=\"Rot13\" method=\"post\">");
21:         out.println("<textarea name=\"text\" ROWS=8 COLS=55>");
22:         out.println(translation);
23:         out.println("</textarea>");
24:         out.println("<p><input type=\"submit\" value=\"translate\">");
25:         out.println("</form>");
26:         out.println("</body>");
27:         out.println("</html>");
28:     }
29:
30:     public void doGet(HttpServletRequest req, HttpServletResponse res)
31:         throws ServletException, IOException {
32:
33:         doPost(req, res);
34:     }
35:
36:     String translate(String input) {
37:         StringBuffer output = new StringBuffer();
38:         if (input != null) {
39:             for (int i = 0; i < input.length(); i++) {
40:                 char inChar = input.charAt(i);
41:                 if ((inChar >= 'A') & (inChar <= 'Z')) {
42:                     inChar += 13;
43:                     if (inChar > 'Z')
44:                         inChar -= 26;
45:                 }
46:                 if ((inChar >= 'a') & (inChar <= 'z')) {
47:                     inChar += 13;
48:                     if (inChar > 'z')
49:                         inChar -= 26;
50:                 }
            }
        }
    }
}
```

```

51:         output.append(inChar);
52:     }
53: }
54:     return output.toString();
55: }
56: }

```

请输入并保存该 servlet，然后使用 Java 编译器对它进行编译。

Rot13 servlet 接收 Web 表单中的文本，使用 ROT-13 对文本进行转换，然后将结果显示在一个新的 Web 表单中。ROT-13 是一个通过字母替代来对文本进行加密的简单方法。每个字母都被替换为它后面的第 13 个字母：A 变成 N、N 变成 A、B 变成 O、O 变成 B、C 变成 P、P 变成 C，以此类推。

由于 ROT-13 加密机制很容易被解码，所以不能用于对重要信息进行加密。它通常被用于诸如 Usenet 新闻组等 Internet 论坛中。例如，电影新闻组中的人要共享一个披露即将上映的电影细节的“探测器”，可使用 ROT-13 对其进行加密，以防人们偶然间看到它。

#### 注意

想知道 1973 年的电影《Soylent Green》(中译名“超世纪大谋杀”，美国著名科幻影片)中的大秘密吗？请将下述 ROT-13 文本进行解码：Fbba gurl'yy or oerrqvaf hf yvxxr pnggyr! Lbh'ir tbq gb jnea rirelbar naq gryy gurz! Fblyrag terra vf znqr bs crbcyr! Lbh'ir tbq gb gryy gurz! Fblyrag terra vf crbcyr!

要使 servlet ROT-13 可用，必须将其类文件发布到指定用于 Java servlet 的 Web 服务器上的文件夹中。

在 Tomcat 中，servlets、其他类和 JSP 页面被放在文件夹 webapps 的子文件夹中。要部署 servlet 的类文件，方法之一是将其存储在 webapps 文件夹层次结构中的子文件夹 WEB-INF/classes 中。

Tomcat 5.5 包含多个 servlet 示例，如果安装 Tomcat 5.5 时选择安装这些示例，它们将位于文件夹 webapps 下的子文件夹 servlets-examples 和 jsp-examples 中。要将 ROT-13 servlet 部署到文件夹 servlet-examples 中，可将 Rot13.class 存储到文件夹 webapps\servlet-examples\WEB-INF\classes (对于 Windows 系统) 或 webapps/servlet-examples/WEB-INF/classes (对于 Linux 系统) 中。

如果将 Rot13.class 放在该文件夹中，请对其父文件夹中的文件 web.xml 进行编辑——添加如下代码行：

```

<servlet>
  <servlet-name>Rot13</servlet-name>
  <servlet-class>Rot13</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Rot13</servlet-name>
  <url-pattern>/servlet/Rot13</url-pattern>
</servlet-mapping>

```

web.xml 文件用于配置 Web 应用程序，如存储在文件夹 webapps\servlet-examples 中的 servlet 示例。上述代码行必须放在开始标记<web-app>和结束标记</web-app>之间。

添加类文件并编辑 web.xml 后，重新启动 Tomcat，并在 Web 浏览器中输入该 servlet 的地址以运行它。

具体使用什么地址取决于该 servlet 存储在 webapps 文件夹中的什么地方。如果使用的是前面的配置，它将位于 /servlets-examples/servlet/Rot13 中，因此应输入地址 <http://localhost:8080/servlets-examples/servlet/Rot13>。

### 21.2.1 使用 cookie

很多网站都可定制，以跟踪有关用户的信息及其想让网站显示的特性。之所以能够进行这种定制

是由于 Web 浏览器有一种名为 cookie 的特性。cookie 是一种小文件，其中包含网站要记录的关于用户的信息，如用户名、访问次数等。这种文件保存在用户的计算机中，网站只能读取它存储在用户系统上的 cookie。

出于隐私方面的考虑，可对大多数 Web 浏览器进行配置，以拒绝所有的 cookie 或询问用户是否允许网站创建 cookie。大多数浏览器的默认行为是接受所有的 cookie。

通过 servlet，您可以很容易地在用户运行您的程序时创建和检索 cookie。cookie 是由 javax.servlet.http 包中的 Cookie 类支持的。

要创建 cookie，可调用构造函数 `Cookie(String, String)`。其中第一个参数是 cookie 的名称，第二个参数是 cookie 的值。

cookie 的用途之一是计算用户加载 servlet 的次数。下面的语句创建一个名为 visits 的 cookie，并将其初始值设置为 1：

```
Cookie visitCookie = new Cookie("visits", "1");
```

创建 cookie 时，必须指定它在用户计算机上的有效期。cookie 的有效期可以是一个小时、一天、一年或任何时间。当 cookie 不再有效时，Web 浏览器将自动地删除它。

要设置 cookie 的有效期（单位为秒），可调用其 `setMaxAge(int)` 方法。如果您使用负数作为参数，则 cookie 只在用户 Web 浏览器关闭前有效；如果使用 0 作为参数，则 cookie 将不会被存储到用户的计算机中。

#### 注意

创建寿命为 0 的 cookie 旨在命令 Web 浏览器删除已有的 cookie。

cookie 随 Web 浏览器显示的数据一起被发送给用户计算机。要发送 cookie，可调用 `HttpServletResponse` 对象的 `addCookie(Cookie)` 方法。

可以将多个 cookie 加入到响应中。cookie 被存储到用户计算机中时，将与创建它的网页或程序的 URL 相关联。可以将多个 cookie 与同一个 URL 相关联。

Web 浏览器请求 URL 时，将检查是否有与该 URL 相关联的 cookie。如果有，将随请求一起被发送出去。

在 servlet 中，可调用 `HttpServletRequest` 对象的 `getCookies()` 方法来接收一个 `Cookie` 对象数组。您可以调用每个 cookie 的 `getName()` 和 `getValue()` 方法来了解 cookie 的信息并对数据进行操作。

程序清单 21.2 的 `ColorServlet` 是 ROT-13 的扩展版本，它让用户能够选择该 servlet 显示的网页的背景色。该颜色被存储为名为 color 的 cookie。每当该 servlet 被加载时，它都将向 Web 浏览器请求该 cookie。

程序清单 21.2 完整的 `ColorServlet.java` 源代码

```
1: import java.io.*;
2:
3: import javax.servlet.*;
4: import javax.servlet.http.*;
5:
6: public class ColorServlet extends HttpServlet {
7:
8:     public void doPost(HttpServletRequest req, HttpServletResponse res)
9:         throws ServletException, IOException {
10:
11:         String pageColor;
12:         String colorParameter = req.getParameter("color");
13:         if (colorParameter != null) {
14:             Cookie colorCookie = new Cookie("color", colorParameter);
15:             colorCookie.setMaxAge(31536000);
16:             res.addCookie(colorCookie);
17:             pageColor = colorParameter;
18:         } else {
```

```

19:         pageColor = retrieveColor(req.getCookies());
20:     }
21:     String text = req.getParameter("text");
22:     String translation = translate(text);
23:     res.setContentType("text/html");
24:     ServletOutputStream out = res.getOutputStream();
25:     out.println("<html>");
26:     out.println("<body bgcolor=\"" + pageColor + "\">");
27:     out.println("<head><title>ROT-13 Translator</title></head>");
28:     out.println("<h1>ROT-13 Translator</h1>");
29:     out.println("<p>Text to translate:</p>");
30:     out.println("<form action=\"" + ColorServlet + "\" method=\"" + post + "\">");
31:     out.println("<input type=\"" + text + "\" name=\"" + color + "\" value=\"" + pageColor + "\" SIZE=40>");
32:     out.println("<input type=\"" + submit + "\" value=\"" + submit + "\">");
33:     out.println("</form>");
34:     out.println("</body>");
35:     out.println("</html>");
41: }
42:
43: public void doGet(HttpServletRequest req, HttpServletResponse res)
44:     throws ServletException, IOException {
45:
46:     doPost(req, res);
47: }
48:
49: String translate(String input) {
50:     StringBuffer output = new StringBuffer();
51:     if (input != null) {
52:         for (int i = 0; i < input.length(); i++) {
53:             char inChar = input.charAt(i);
54:             if ((inChar >= 'A') & (inChar <= 'Z')) {
55:                 inChar += 13;
56:                 if (inChar > 'Z')
57:                     inChar -= 26;
58:             }
59:             if ((inChar >= 'a') & (inChar <= 'z')) {
60:                 inChar += 13;
61:                 if (inChar > 'z')
62:                     inChar -= 26;
63:             }
64:             output.append(inChar);
65:         }
66:     }
67:     return output.toString();
68: }
69:
70: String retrieveColor(Cookie[] cookies) {
71:     String inColor = "#FFFFFF";
72:     for (int i = 0; i < cookies.length; i++) {
73:         String cookieName = cookies[i].getName();
74:         if (cookieName.equals("color")) {
75:             inColor = cookies[i].getValue();
76:         }
77:     }
78:     return inColor;
79: }
80: }

```

图 21.2 所示是该 servlet 在 Web 浏览器中的运行情况。

要修改该页面的背景色，可在文本框 Choose a new color 中输入新值并单击 Change Color 按钮。

要修改页面的背景色，可在文本框 Background Color of Page 中输入新值并单击 Submit 按钮。

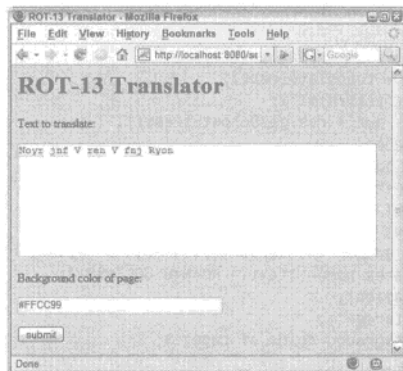


图 21.2 ColorServlet 生成的网页

颜色是用#和 3 个两位的十六进制数表示的（在图 21.2 中，这些数字分别为 FF、CC 和 99）。这些数字表示红色、绿色和蓝色的量，取值范围为 00~FF。如果您不熟悉十六进制颜色，可在测试该 servlet 时使用下述颜色。

- #FF0000：亮红色。
- #00FF00：亮绿色。
- #0000FF：亮蓝色。
- #FFAAAA：淡红色。
- #AAFFAA：淡绿色。
- #AAAAFF：淡蓝色。
- #FFCC66：奶黄色。

### 21.2.2 使用会话

servlet 最强大的特性之一是支持会话（sessions）——一种在 servlet 被使用期间监视用户的手段。

通常，Web 是一种无状态的协议，这意味着没有简单的方法可用于跟踪用户在页面间导航的情况。Web 浏览器向服务器请求 URL 时，将收到相应的文件，随后服务器将这些忘得一干二净，而没有采取措施来跟踪用户在一段时间内对网站执行的操作。

如果仅提供一系列静态页面，上述信息并不重要。但对很多 Web 应用程序而言，这些信息是不可缺少的，尤其是在电子商务中：在结账前，网上商店需要知道用户将哪些商品加入到了购物车中，而在使用信用卡结账时，需要知道顾客所购商品的总金额，等等。

通过使用表示会话的类 HttpSession，servlet 能够保存用户的状态。对于每个运行 servlet 的用户，可以提供一个会话对象。

创建、删除和维护会话的工作是由运行 servlet 的服务器在幕后完成的。

可以通过调用 servlet 的请求对象的 getSession(Boolean)来创建或检索用户的会话。如果需要在用户没有会话时创建一个，则应指定参数 true，如下例所示：

```
HttpSession state = req.getSession(true);
```

调用相应对象的方法对 servlet 的输出进行排列之前，必须以上述方式访问会话对象。

可以通过调用方法 isNew()来判断会话对象是否是新创建的。如果是，该方法将返回 true。

如果需要在用户使用 servlet 时对某些内容进行记录，可以将它们存储在会话对象中。

会话对象存储对象的方式类似于第 8 章中介绍的 Vector。

存储在会话中的对象被称为会话的属性。要设置属性，可调用会话的 `setAttribute(String, Object)` 方法，并提供两个参数：属性名和对象。

要检索属性，可调用 `getAttribute(String)` 方法，并将属性名作为参数。该方法返回一个对象，必须将其转换为相应的类对象。如果没有指定的属性，该方法返回 `null`。

当不再属性需要时，可调用方法 `removeAttribute(String)` 来删除它，并将属性名作为参数。如果没有指定的属性，该方法不返回 `null`，而是不执行任何操作。

如果会话无效，上述 3 个方法都将引发 `IllegalStateException` 异常。导致这种情况的原因有：服务器已将会话删除或者由于某种错误导致无法保留会话。

接下来的项目使用会话来记录用户是否向 servlet 提供了登录信息，如图 21.3 所示。

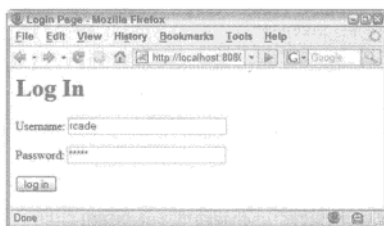


图 21.3 使用 servlet 来要求用户登录

让用户登录的 servlet 验证其用户名和密码。可能的情况有下列 3 种：

- 用户登录前，servlet 已经运行。在这种情况下，必须显示一个表单，让用户能够输入用户名和密码。
- 用户登录时运行 servlet。必须采取某种方法对用户提供的用户名和密码进行验证，例如将其同数据库进行比较。
- 用户登录后运行 servlet。

在上述 3 种情况下，都需要知道之前发生的情况，因此需要使用会话。

应用程序 `LoginServlet` 使用 3 个会话属性来处理用户登录：用户名、密码和 `loggedIn`。其中 `loggedIn` 是一个 `Boolean` 对象，如果用户已经登录，其值为 `true`；否则为 `false`。程序清单 21.3 列出了该应用程序的源代码。

程序清单 21.3 完整的 `LoginServlet.java` 源代码

```
1: import java.io.*;
2: import java.util.Date;
3: import javax.servlet.*;
4: import javax.servlet.http.*;
5:
6: public class LoginServlet extends HttpServlet {
7:
8:     public void doPost(HttpServletRequest req, HttpServletResponse res)
9:         throws ServletException, IOException {
10:
11:         HttpSession session = req.getSession();
12:         Boolean loggedIn = (Boolean) session.getAttribute("loggedIn");
13:         if (loggedIn == null) {
14:             loggedIn = new Boolean(false);
15:         }
16:         String username = req.getParameter("username");
17:         String password = req.getParameter("password");
18:         Date lastVisit;
19:         res.setContentType("text/html");
20:         ServletOutputStream out = res.getOutputStream();
21:         out.println("<html>");
22:         out.println("<body>");
23:         out.println("<head><title>Login Page</title></head>");
```

```

24:         if (loggedIn.booleanValue() == true) {
25:             // user is already logged in
26:             username = (String) session.getAttribute("username");
27:             password = (String) session.getAttribute("password");
28:             lastVisit = (Date) session.getAttribute("lastVisit");
29:             out.println("<p>Welcome back, " + username);
30:             out.println("<p>You last visited on " + lastVisit);
31:             lastVisit = new Date();
32:             session.setAttribute("lastVisit", lastVisit);
33:         } else {
34:             if (username == null) {
35:                 // user has not submitted the form required to log in
36:                 out.println("<h1>Log In</h1>");
37:                 out.println("<form action=\"LoginServlet\" " +
38:                     "method=\"post\">");
39:                 out.println("<p>Username:");
40:                 out.println("<input type=\"text\" name=\"username\" " +
41:                     "value=\"\" SIZE=30>");
42:                 out.println("<p>Password:");
43:                 out.println("<input type=\"password\" name=\"password\" " +
44:                     "value=\"\" SIZE=30>");
45:                 out.println("<p><input type=\"submit\" value=\"log in\">");
46:                 out.println("</form>");
47:             } else {
48:                 // user has submitted the login form
49:                 out.println("Logging in " + username);
50:                 session.setAttribute("username", username);
51:                 session.setAttribute("password", password);
52:                 session.setAttribute("loggedIn", new Boolean(true));
53:                 session.setAttribute("lastVisit", new Date());
54:                 out.println("<a href=\"LoginServlet\">Reload Page</a>");
55:             }
56:             out.println("</body>");
57:             out.println("</html>");
58:         }
59:     }
60:
61:     public void doGet(HttpServletRequest req, HttpServletResponse res)
62:         throws ServletException, IOException {
63:
64:         doPost(req, res);
65:     }
66: }

```

在 Web 浏览器中首次加载该 servlet 时，它将显示一个如图 21.3 所示的表单。

用户填写好表单并单击 Submit 按钮后，将显示一个网页，其中包含文本“Logging in”和超链接 Reload Page。

用户单击该超链接后，将出现一个包含问候语的页面，其中的问候语与下面类似：

Welcome back, rcade

You last visited on Sat Feb 29 18:04:45 EST 2007

该 servlet 不包含任何对提供的用户名和密码进行检查的代码，而只是将它们存储在会话对象中，这样以后该 servlet 再次运行时便可以使用它们。

## 21.3 JSP

Java servlet 使得容易动态地生成 HTML 文本，生成能够根据用户输入而改变的页面。

然而，servlet 使得难以生成始终不变的 HTML 文本，因为使用 Java 语句来输出 HTML 非常麻烦且乏味。

要修改 HTML，Java 程序员需要对 servlet 进行处理。必须对 servlet 进行编辑、重新编译，并将其发布到 Web 上，很少组织愿意将这种任务交给非程序员去处理。

JSP 是 servlet 的补充，而不是替代品。它们使得容易将如下两种 Web 内容分开。

- 静态内容：网页中不变的部分，如网上商店中关于商品的介绍。
- 动态内容：servlet 生成的网页部分，如商品的价格和供货情况，这些数据可能随商品的销售而变化。

如果只使用 servlet，则进行细微的修改（如更正输入错误、重写段落或修改 HTML 标记以改变网页的呈现方式）将极其困难。要进行任何修改，都必须对 servlet 进行编辑、编译、测试，并将其重新部署到 Web 服务器中。

使用 JSP，可以将网页的静态内容放到 HTML 文档中，并在其中调用 servlet。您还可以在网页中使用 Java 语言的其他元素，如表达式、if-then 语句块和变量。支持 Tomcat 规范的 Web 服务器知道如何读取这些网页、执行其中的 Java 代码、生成 HTML 文档，就好像整个任务是由 servlet 处理的一样。实际上，JSP 使用 servlet 来完成所有的工作。

创建 JSP 页面的方式与创建 HTML 文档相同：使用文本编辑器或 Web 发布程序（如 Macromedia Dreamweaver）。保存页面时，使用文件扩展名.jsp，以指出这是 JSP 页面，而不是 HTML 文档。然后便可以在 Web 服务器上发布该页面，就像发布 HTML 文档一样——只要服务器支持 servlet 和 JSP。

用户首次请求 JSP 页面时，Web 服务器将合成一个新的 servlet，用于显示该页面。该 servlet 将页面中的所有内容组合起来：

- 用 HTML 标记的文本；
- 对 Java servlet 的调用；
- Java 表达式和语句；
- 特殊的 JSP 变量。

### 21.3.1 编写 JSP

JSP 页面由 3 种元素构成，这些元素都有其特殊的类似于 HTML 的标记。

- Scriptlet：网页被加载时执行的 Java 语句，这样的语句都位于标记`<%和%>`之间。
- 表达式：Java 表达式，生成的输出被显示在网页中，表达式位于标记`<%=和%>`之间。
- 声明：用于创建实例变量以及处理显示页面所需的设置任务的语句。它们位于标记`<%!和%>`之间。

#### 1. 使用表达式

程序清单 21.4 中的 JSP 页面包含一个表达式——对构造函数 `java.util.Date()` 的调用。这个构造函数生成一个字符串，其中包含当前的时间和日期。请使用能够将文件保存为纯文本格式的文本编辑器输入该程序（用于创建 Java 源代码的编辑器也可实现这种目标）。

程序清单 21.4 完整的 time.jsp 源代码

```
1: <html>
2: <head>
3: <title>Clock</title>
4: </head>
5: <body>
6: <h1 align="Center">
7: <%= new java.util.Date() %>
8: </h1>
9: </body>
10: </html>
```

保存该文件后，将其上传到 Web 服务器中其他网页所在的文件夹中。与 Java servlet 不同，JSP 可放在网上任何可访问的文件夹中，而无需放在 servlet 专用的文件夹中。



在 Tomcat 5.5 中, 可以将该页面存储在文件夹 webapps 中的任何地方。如果存储在文件夹 webapps\jsp-examples 中, 则加载它时应输入地址 `http://localhost:8080/jsp-examples/time.jsp`。

当您首次加载该页面的 URL 时, Web 服务器将自动把这个 JSP 合成为 servlet。这将降低页面的加载速度, 但响应后续请求的速度将快得多。

time.jsp 的输出如图 21.4 所示。

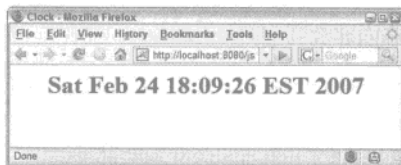


图 21.4 在 JSP 页面中使用表达式

当 JSP 页面包含表达式时, 表达式将被计算, 得到的值被显示在页面上。如果每次运行时, 表达式的结果都不同, 就像 time.jsp 中的第 7 行那样, 则当它被加载到 Web 浏览器中时, 将在页面中反映出来。

在 JSP 页面的表达式和其他元素中, 可以使用下列变量名引用多个 servlet 对象。

- out: servlet 输出流。
- request: HTTP servlet 请求。
- response: HTTP servlet 响应。
- session: 当前 HTTP 会话。
- application: 用于同 Web 服务器进行通信的 servlet 场境 (context)。
- config: 用于查看 servlet 如何被初始化的 servlet 配置对象。

在页面中, 使用这些变量可以调用可在 servlet 中调用的方法。

接下来的页面 environment.jsp 演示了如何使用 request 变量。这个变量表示一个 `HttpServletRequest` 对象, 您可以调用该对象的方法 `getHeader(String)` 来获得 HTTP 报头, 该报头更详细地描述了请求。程序清单 21.5 列出了 environment.jsp 的源代码。

程序清单 21.5 完整的 environment.jsp 源代码

```

1: <html>
2: <head>
3: <title>Environment Variables</title>
4: </head>
5: <body>
6: <ul>
7: <li>Accept: <%= request.getHeader("Accept") %>
8: <li>Accept-Encoding: <%= request.getHeader("Accept-Encoding") %>
9: <li>Connection: <%= request.getHeader("Connection") %>
10: <li>Content-Length: <%= request.getHeader("Content-Length") %>
11: <li>Content-Type: <%= request.getHeader("Content-Type") %>
12: <li>Cookie: <%= request.getHeader("Cookie") %>
13: <li>Host: <%= request.getHeader("Host") %>
14: <li>Referer: <%= request.getHeader("Referer") %>
15: <li>User-Agent: <%= request.getHeader("User-Agent") %>
16: </ul>
17: </body>
18: </html>

```

在第 7~15 行中, 每行都包含了一个 `getHeader()` 调用, 它们检索 HTTP 请求的各个报头。输出如图 21.5 所示。每个报头的值取决于 Web 服务器和您使用的 Web 浏览器, 因此您看到的 User-Agent、Referer 和其他报头的值可能与此不同。

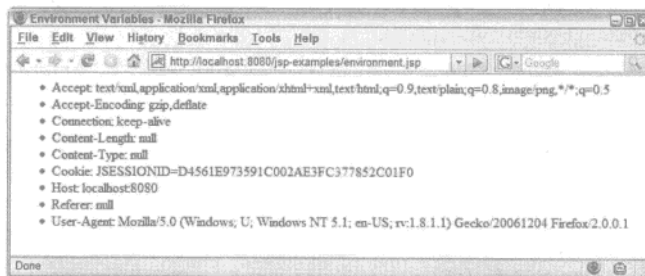


图 21.5 在 JSP 页面中使用 servlet 变量

## 2. 使用 Scriptlet

在 JSP 中, 还可以使用 Java 语句: 调用方法、给变量赋值、创建条件语句等。这些语句以标记<%打头, 以标记%>结束。这些标记中可以包含多条语句。

JSP 页面中的语句被称为 scriptlet。可以使用任何可用于表达式中的 servlet 变量。

程序清单 21.6 列出了 shop-for-books.jsp 的代码, 该网页显示网上书店的图书列表, 对于每本图书, 都有一个超链接连接到该图书的页面。

程序清单 21.6 完整的 shop-for-books.jsp 源文件

```

1: <html>
2: <head>
3: <title>Shop for Books</title>
4: </head>
5: <body>
6: <h2 align="Left">Favorite Books</h2>
7: <%
8: String[] bookTitle = { "Catch-22", "Something Happened",
9:   "Good as Gold" };
10: String[] isbn = { "0684833395", "0684841215", "0684839741" };
11: String amazonLink = "http://www.amazon.com/exec/obidos/ASIN/";
12: String bnLink = "http://search.barnesandnoble.com/booksearch/"
13:   + "isbnInquiry.asp?isbn=";
14:
15: String store = request.getParameter("store");
16: if (store == null) {
17:   store = "Amazon";
18: }
19: for (int i = 0; i < bookTitle.length; i++) {
20:   if (store.equals("Amazon"))
21:     out.println("<li><a href=\"" + amazonLink + isbn[i] + "\">" +
22:       bookTitle[i] + "</a>");
23:   else
24:     out.println("<li><a href=\"" + bnLink + isbn[i] + "\">" +
25:       bookTitle[i] + "</a>");
26: }
27: %>
28: <p>Preferred Bookstore:
29: <form action="shop-for-books.jsp" method="POST">
30: <p><input type="radio" value="Amazon"
31: <%= (store.equals("Amazon")) ? " checked" : "" %>
32: name="store"> Amazon.Com
33: <p><input type="radio" value="BN"
34: <%= (store.equals("BN")) ? " checked" : "" %>
35: name="store"> Barnes & Noble
36: <p><input type="submit" value="Change Store">
37: </form>
38: </body>
39: </html>

```

该页面底部包含一个表单, 让用户挑选喜欢的书店来进行网上购物。

第 29 行将表单提交给页面的 URL。因为页面实际上是 servlet，因此它们也可以接收 POST 或 GET 发送的表单数据。

如果用户选择的是网站 Amazon.com，则该页面将使用 store 来存放“Amazon”；如果用户选择的是 Barnes & Noble，则存储“BN”。

测试该服务器页面时，应注意的是，表单上的单选按钮总是与选定的商店匹配。其原因在于第 29 和 31 行的表达式。下面是其中的一个：

```
<%= (store.equals("Amazon") ? " checked" : "") %>
```

该表达式在三目运算符中使用了条件 store.equals(“Amazon”)。如果该条件为真，则整个表达式的结果为 checked，否则，为空字符串(“”)。

该表达式的值作为 JSP 页面的一部分被显示出来。图 21.6 显示该页面在 Web 浏览器中的情况。

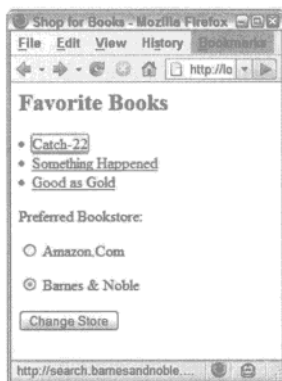


图 21.6 使用 scriptlet 显示动态内容

### 3. 使用声明

可以插入到 JSP 页面中的最后一个元素是声明——用于设置变量或方法的语句，当页面被编译为 servlet 时，将定义这些变量和方法。

声明位于标记<%!和%>之间，正如下例所示：

```
<!% boolean noCookie = true %>
<!% String userName = "New user" %>
```

上述声明创建了两个实例变量：noCookie 和 userName。当该 JSP 页面被编译为 servlet 时，这些变量将是类定义的一部分。

程序清单 21.7 中的 JSP 页面使用声明来显示一个计数器。

#### 程序清单 21.7 完整的 counter.jsp 源代码

```
1: <%@ page import="counter.*" %>
2: <html>
3: <head>
4: <title>Counter Example</title>
5: </head>
6: <body>
7: <h1>JSP Stats</h1>
8: <%= Counter visits; %>
9: <%= int count; %>
10:
11: <%
12: visits = new Counter(application.getRealPath("counter.dat"));
13: count = visits.getCount() + 1;
14: %>
```

```

15:
16: <p>This page has been loaded <%= count %> times.
17:
18: <% visits.setCount(count); %>
19: </body>
20: </html>

```

运行该页面之前，必须创建一个助手类，它被第 8、12、13 和 18 行的语句调用。  
程序清单 21.8 中的 Counter 类表示一个 Web 计数器，它计算页面被点击的次数。

#### 程序清单 21.8 完整的 Counter.java 源代码

```

1: package counter;
2:
3: import java.io.*;
4: import java.util.*;
5:
6: public class Counter {
7:     private int count;
8:     private String filepath;
9:
10:    public Counter(String inFilepath) {
11:        count = 0;
12:        filepath = inFilepath;
13:    }
14:
15:    public int getCount() {
16:        try {
17:            File countFile = new File(filepath);
18:            FileReader file = new FileReader(countFile);
19:            BufferedReader buff = new BufferedReader(file);
20:            String current = buff.readLine();
21:            count = Integer.parseInt(current);
22:            buff.close();
23:        } catch (IOException e) {
24:            // do nothing
25:        } catch (NumberFormatException nfe) {
26:            // do nothing
27:        }
28:        return count;
29:    }
30:
31:    public void setCount(int newCount) {
32:        count = newCount;
33:        try {
34:            File countFile = new File(filepath);
35:            FileWriter file = new FileWriter(countFile);
36:            BufferedWriter buff = new BufferedWriter(file);
37:            String output = "" + newCount;
38:            buff.write(output, 0, output.length());
39:            buff.close();
40:        } catch (IOException e) {
41:            // do nothing
42:        }
43:    }
44: }

```

编译这个类后，将其保存到 counter.jsp 所在文件夹层次结构中的子文件夹 WEB-INF\classes\counter 中。例如，如果 JSP 页面存储在 webapps\examples\jsp 中，应将类文件存储在 webapps\examples\WEB-INF\classes\counter 中。

Counter 类从文件 counter.dat 中加载一个整数值，该文件存储在 Web 服务器上。getCount() 方法检索计数器的当前值，而方法 setCount(int) 设置计数器的当前值。这个值被设置后，将被保存到文件中，这样计数器将不断增加。

图 21.7 显示了在 Web 浏览器中加载 counter.jsp 时得到的结果。



图 21.7 使用 JSP 和 Java 对象计算网页的点击次数

### 21.3.2 创建 Web 应用程序

结合使用 Java 类、servlet 和 JSP，可以创建交互式 Web 应用程序：以复杂、内聚的方式动态地生成内容以响应用户输入的网站。

每当您在电子商务网站（如 Amazon.com）上购物或使用在线帮助（如 Internet Movie Database, IMDB）时，都在运行 Web 应用程序。

为了解 Java 技术的多个方面是如何在 Web 上协同工作的，将创建 Web 应用程序 Guestbook，它让访问者能够给网站创建者留言。

Guestbook 由如下 3 部分构成。

- guestbook.jsp：一个 JSP 页面，它显示 Web 服务器上一个文本文件中的用户留言并提供一个表单，访问者可通过该表单添加留言。
- guestbookpost.jsp：一个 JSP 页面，它将新的留言条目保存到文本文件中。
- Guestbook.java：一个类，用于将留言保存到留言簿中之前过滤掉其中的一些字符。

该项目中的 JSP 页面使用了大量的 scriptlet 和表达式。程序清单 21.9 列出了 guestbook.jsp 的源代码。

程序清单 21.9 完整的 guestbook.jsp 源代码

```

1: <%@ page import="java.util.*,java.io.*" %>
2: <html>
3: <head>
4: <title>Visitors Who Signed our Guestbook</title>
5: </head>
6: <body>
7: <h3>Visitors Who Signed our Guestbook</h3>
8: <%
9: String id = request.getParameter("id");
10: boolean noSignatures = true;
11: try {
12:     String filename = application.getRealPath(id + ".gbf");
13:     FileReader file = new FileReader(filename);
14:     BufferedReader buff = new BufferedReader(file);
15:     boolean eof = false;
16:     while (!eof) {
17:         String entry = buff.readLine();
18:         if (entry == null)
19:             eof = true;
20:         else {
21:             StringTokenizer entryData = new StringTokenizer(entry, "");
22:             String name = (String) entryData.nextElement();
23:             String email = (String) entryData.nextElement();
24:             String url = (String) entryData.nextElement();
25:             String entryDate = (String) entryData.nextElement();
26:             String ip = (String) entryData.nextElement();
27:             String comments = (String) entryData.nextElement();
28:             out.print("<p>From: " + name);

```

```

29:         if (!email.equals("None"))
30:             out.println(" <" + email + "><br>");
31:         else
32:             out.println("<br>");
33:         if (!url.equals("None"))
34:             out.println("Home Page: <a href=\"\" + url + "\">" +
35:                 url + "</a><br>");
36:         out.println("Date: " + entryDate + "<br>");
37:         out.println("IP: " + ip);
38:         out.println("<blockquote>");
39:         out.println("<p>" + comments);
40:         out.println("</blockquote>");
41:         noSignatures = false;
42:     }
43: }
44: buff.close();
45: } catch (IOException e) {
46:     out.println("<p>This guestbook could not be read because of an error.");
47:     log("Guestbook Error: " + e.toString());
48: }
49: if (noSignatures)
50:     out.println("<p>No one has signed our guestbook yet.");
51: %>
52: <h3>Sign Our Guestbook</h3>
53: <form method="post" action="guestbook-post.jsp">
54:     <table border="0" cellpadding="5" cellspacing="0" width="100%">
55:         <tr>
56:             <td width="15%" valign="top" align="right">Your Name:</td>
57:             <td width="85%"><input type="text" name="name" size="40"></td>
58:         </tr>
59:         <tr>
60:             <td width="15%" valign="top" align="right">Your E-mail Address:</td>
61:             <td width="85%"><input type="text" name="email" size="40"></td>
62:         </tr>
63:         <tr>
64:             <td width="15%" valign="top" align="right">Your Home Page:</td>
65:             <td width="85%"><input type="text" name="url" size="40"></td>
66:         </tr>
67:         <tr>
68:             <td width="15%" valign="top" align="right">Your Comments:</td>
69:             <td width="85%">
70:                 <textarea rows="6" name="comments" cols="40"></textarea>
71:             </td>
72:         </tr>
73:     </table>
74:     <p align="center"><input type="submit" value="Submit" name="B1">
75:     <input type="reset" value="Reset" name="Reset"></p>
76:     <input type="hidden" name="id" value="% id %">
77: </form>
78: </body>
79: </html>

```

保存该页面后，将它存储到服务器中可存储 JSP 页面的任何文件夹中。您甚至可以在完成应用程序的其他工作之前，对该页面进行测试，只要有一个空的留言簿文件。

要创建留言簿文件，可将一个空的文本文件保存到系统中，并将其命名为 `cinema.gbf`。将它存储到与 `guestbook.jsp` 所在位置对应的文件夹中。例如，如果 JSP 页面为 `webapps\jspexamples\guestbook.jsp`，则将该文本文件存储在 `webapps\jspexamples` 中。

加载该 JSP 页面时，必须包含一个参数，以指定要加载的留言簿的 ID，如下面的 URL 所示：

`http://localhost:8080/jsp-examples/guestbook.jsp?id=cinema`

服务器名和文件夹名取决于 `guestbook.jsp` 的发布位置。

图 21.8 是编译 JSP 页面后试图显示文件 `cinema.gbf` 的内容时得到的留言簿。

在这个留言簿文件中，每条留言占一行，并用 ^ 将字段分开。访问者可以提供其姓名、电子邮件地址、主页地址和评论。留言中还包含另外两项内容：留言的日期和时间以及访问者的 IP 地址。

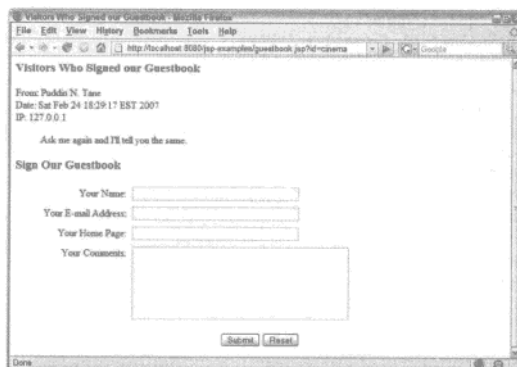


图 21.8 测试 JSP 页面 guestbook.jsp

下面的留言簿示例文件包含两条留言：

Puddin N. Tane^puddin@example.com^None^Sat Feb 24 18:29:17 EST  
2007^127.0.0.1^Ask me again and I'll tell you the same.

接下来要创建的 JSP 页面是 guestbook-post.jsp，它使用访问者提交的新留言更新留言簿。程序清单 21.10 列出了该页面的源代码。

#### 程序清单 21.10 完整的 guestbook-post.jsp 源代码

```
1: <%@ page import="java.util.*,java.io.*,example.*" %>
2: <html>
3: <head>
4: <title>Thank You For Signing Our Guestbook</title>
5: </head>
6: <body>
7: <h3>Thank You For Signing Our Guestbook</h3>
8: <%
9: String id = request.getParameter("id");
10: String[] entryFields = { "name", "email", "url", "comments" };
11: String[] entry = new String[4];
12: for (int i = 0; i < entryFields.length; i++) {
13:     entry[i] = Guestbook.filterString(request.getParameter(entryFields[i]));
14: }
15: Date now = new Date();
16: String entryDate = now.toString();
17: String ip = request.getRemoteAddr();
18: %>
19:
20: <p>Your entry looks like this:
21: <p>From: <%= entry[0] %>
22: <%= (entry[1].equals("None") ? "<"+entry[1]+>" : "") %><br>
23: <% if (entry[2].equals("None")) { %>
24: Home Page: <a href="<%= entry[2] %>"><%= entry[2] %></a><br>
25: <% } %>
26: Date: <%= entryDate %><br>
27: IP: <%= ip %>
28: <blockquote>
29: <p><%= entry[3] %>
30: </blockquote>
31:
32: <%
33: try {
34:     boolean append = true;
35:     String filename = application.getRealPath(id + ".gbf");
36:     FileWriter fw = new FileWriter(filename, append);
37:     BufferedWriter fileOut = new BufferedWriter(fw);
38:     String newEntry = entry[0] + "" + entry[1] + "" + entry[2] + ""
39:         + entryDate + "" + ip + "" + entry[3];
40:     fileOut.write(newEntry, 0, newEntry.length());
41:     fileOut.newLine();
42:     fileOut.close();
```

```

43: } catch (IOException e) {
44:     out.println("<p>This guestbook could not be updated.");
45:     log("Guestbook Error: " + e.toString());
46: }
47: %>
48:
49: <p><a href="guestbook.jsp?id=<%= id %>">View the Guestbook</a>
50: </body>
51:
52: </html>51: </html>

```

页面 `guestbook-post.jsp` 收集 Web 表单中的数据，删除数据中不能加入到留言簿中的字符，然后将结果保存到一个文本文件中。

每个留言簿都存储在一个独立的文件中，该文件的名称为留言簿的 ID 参数，扩展名为 `.gbf`。如果留言簿的 ID 为 `cinema`，则文件名为 `cinema.gbf`。

与该 Web 应用程序中的另一个 JSP 页面一样，`guestbook-post.jsp` 也可以被存放到保存 JSP 页面的 Web 服务器的任何文件夹中。这里将该页面存储到 `guestbook.jsp` 和 `cinema.gbf` 所在的文件夹中。

运行应用程序 `Guestbook` 之前，必须创建一个 Java 类，用于将留言中的一些文本过滤掉。

由于留言在文件中的存储方式，留言簿中不能包含下述 3 种字符：

- `^`，
- 回车符，在 Java 中，它对应整数值 13；
- 换行符，它对应整数值 10。

为删除这些字符，将创建一个名为 `Guestbook` 的助手类。这个类有一个名为 `filterString(String)` 的静态方法，它将字符串中的这 3 种字符删除。

程序清单 21.11 列出了这个类的源代码。

程序清单 21.11 完整的 `Guestbook.java` 源代码

```

1: package example;
2:
3: public class Guestbook {
4:     public static String filterString(String input) {
5:         input = replaceText(input, '^', ' ');
6:         input = replaceText(input, (char)13, ' ');
7:         input = replaceText(input, (char)10, ' ');
8:         return input;
9:     }
10:
11:     private static String replaceText(String inString, char oldChar,
12:         char newChar) {
13:
14:         while (inString.indexOf(oldChar) != -1) {
15:             int oldPosition = inString.indexOf(oldChar);
16:             StringBuffer data = new StringBuffer(inString);
17:             data.setCharAt(oldPosition, newChar);
18:             inString = data.toString();
19:         }
20:         return inString;
21:     }
22: }

```

这个类的大部分工作是由第 11~21 行的 `replaceText()` 方法完成的。它接受 3 个参数：

- 一个字符串，其中可能包含不想要的字符；
- 要删除的字符；
- 替换被删除字符的字符。

编译 `Guestbook` 类后，将它放到与 JSP 页面存储位置对应的子文件夹 `WEB-INF\classes\examples`。例如，如果 JSP 页面存储在 `webapps\jspexamples\jsp` 中，则应将类文件存储在 `webapps\jspexamples\`



WEB-INF\classes\examples 中。

要测试页面 guestbook-post.jsp, 可再次使用 ID 参数 cinema 打开显示留言的页面, 如下例所示:

`http://localhost:8080/java/guestbook.jsp?id=cinema`

添加几条留言, 看看它们在留言簿中是如何显示的。

## 21.4 JSP 标准标记库

对于希望将其技能用于万维网上的 Java 程序员来说, Java 语言在服务器端编程方面的进展无疑属于“不知道走就想跑”。

当 servlet 面世时, 它使得可以使用 Java 来创建类似于 CGI 脚本的程序, 以收集 Web 表单中的输入和 URL 参数, 进而生成 HTML 进行响应。对于编写处理邮件或其他简单任务的程序而言, 这种技术很不错, 但由于大型 Web 应用程序包含多个页面, 如果使用 Java 语句来生成 HTML 输出, 将非常繁琐。另外, 修订起来也更困难, 尤其是在非程序员参与其中时。

两年后, Sun 沿正确的方向再次迈出了一步: 推出了 JSP, 它使得很容易将静态 HTML 输出和 Java 语句生成的动态输出组合起来。通过使用 JSP, 可以在网页的 HTML 标记中放置 Java 代码, 并可以像编辑其他页面那样编辑这种页面。程序员还可以创建自定义的 JSP 标记, 用于同 Java 对象进行交互。这种页面被自动编译为 servlet。

不幸的是, 实践证明, 允许在 JSP 中使用 Java 代码是错误的, 因为它纵容了这样一种坏习惯: 将大量关键任务应用程序代码放在难以维护、不安全、容易被编辑 HTML 的人破坏的地方。例如, 创建、打开和查询数据库连接, 并使用用户名和密码访问数据的代码可能被放置在 JSP 页面中。

JSP 标准标记库 (Standard Tag Library, JSTL) 的发布向前迈出了一大步。JSTL 是一组自定义的 JSP 标记和一种新的数据访问语言, 让基于 JSP 的 Web 应用程序无需借助于 Java 代码便能够处理内容显示工作。

标记库也被称为 taglibs, 由可以像 HTML 标记那样放置到 JSP 页面中的标记组成。JSTL 的标记库提供了大多数 Web 应用程序都需要的功能。有功能与循环和条件语句相同的标记、使用结构化查询语言 (SQL) 读取可扩展标记语言 (XML) 文档和数据库结果集的标记、访问 JavaBeans 的标记以及支持其他标记库的标记。

JSTL 有 2 个互补的部分组成。

- 5 个自定义的 JSP 标记库: 提供 Web 应用程序所需的核心功能。
- JSTL 表达式语言: 使得无需使用 JSP 表达式或 Java 语句便可以在运行阶段存取数据。

每个 JSTL 库还有一个不同的版本, 它们采用原来的 JSP 语法来使用 Java 表达式。

程序清单 21.12 中的 hello.jsp 演示了如何使用 JSTL 和表达式语言, 它是一个 JSP 页面, 将一个名为 name 的参数作为问候语的一部分。

程序清单 21.12 完整的 hello.jsp 源代码

```
1: <html>
2: <head>
3:   <title>Hello Example</title>
4: </head>
5:
6: <body>
7: <%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>
8:
9: <c:choose>
10:   <c:when test='${!empty param.name}'>
11:     <h2>Hello, <c:out value='${param.name}' /></h2>
12:   </c:when>
13:   <c:otherwise>
14:     <h2>Hello, stranger</h2>
```

```

15:     </c:otherwise>
16: </c:choose>
17:
18: </body>
19: </html>

```

页面 `hello.jsp` 查找页面地址 (URL) 中指定的参数 `name`。例如, URL `http://example.com/hello.jsp?name=Sailor` 将参数 `name` 的值设置为 `Sailor`, 导致该页面显示问候语 `Hello Sailor`。如果参数 `name` 被省略, 将显示问候语 `Hello Stranger`。

页面中的第一段 JSP 代码是如下伪指令 (directive):

```
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>
```

和其他标记库一样, 在页面中使用标记之前, 必须使用伪指令使相应的 JSTL 库可用。上述伪指令使得可以在页面中使用核心库。对于这个库中的所有标记, 都必须在标记名前加上字符 “c:”, 如下例所示:

```
<c:out value='${param.name}' />
```

JSTL 标记被称为操作 (action), 因为它们生成动态的 Web 内容。和 XML 元素一样, 操作可以是独立的 (`<tagname>`), 也可以是成对的 (`<tagname>...</tagname>`)。

操作 `c:out` 显示其 `value` 属性指定的局部变量或实例变量的内容。

变量是使用 JSTL 的表达式语言指定的, 后者是从 ECMAScript (JavaScript) 和 XPath 那里借鉴而来的一种数据存取语法。使用表达式语言的语句被放在 “\${” 和 “}” 之间。

在前面的例子中, `param` 是几个标准表达式语言变量之一, 它包含有关页面、Web 应用程序和 servlet 容器的信息。变量 `param` 是一个集合, 存储了所有的页面参数, 其中每个参数都被表示为字符串。

该页面中余下的内容是 3 个核心操作, 用于创建一个条件语句块:

```

<c:choose>
  <c:when test='${!empty param.name}'>
    <h2>Hello, <c:out value='${param.name}' /></h2>
  </c:when>
  <c:otherwise>
    <h2>Hello, stranger</h2>
  </c:otherwise>
</c:choose>

```

语句块 `c:choose-c:when-c:otherwise` 的功能与 Java 语句 `switch-case-default` 相同, 它仅在第一个 `c:when` 操作有一个值为 `true` 的 `test` 属性时, 才显示其中的 HTML 输出。如果没有任何操作为 `true`, 将显示 `c:otherwise` 的内容。

JSTL 由 5 个标记库组成。

- 核心库: 使用前缀 `c`, 默认 URI 为 `http://java.sun.com/jsp/jstl/core`。它提供常规特性: 输出、有条件地显示内容、循环、在多种作用域中创建变量、访问 `JavaBeans`、异常处理、URL 导入和 URL 重定向。
- SQL 库: 前缀为 `sql`, 默认 URI 为 `http://java.sun.com/jsp/jstl/sql`。它提供数据库访问功能: 选择数据源、查询、更新、事务和遍历结果集。
- 国际化和格式化库: 前缀为 `fmt`, 默认 URI 为 `http://java.sun.com/jsp/jstl/fmt`。它提供如下操作: 地区和资源绑定、文本本地化以及数字和日期的格式化。
- XML 处理库: 前缀为 `x`, 默认 URI 为 `http://java.sun.com/jsp/jstl/xml`。它用于支持 XML: 分析、XPath 访问、XSLT 转换、遍历节点和有条件地处理。
- 函数库: 前缀为 `fn`, 默认 URI 为 `http://java.sun.com/jsp/jstl/functions`。它提供用于操纵字符串和集合的函数。

表达式语言还提供了用于从实例变量、数据结构和数组或链表中检索信息的运算符: `${object.varName}`、`${object["name"]}` 和 `${object[1]}`。

另外, 还有算术运算符 (+、-、\*、/、%)、比较运算符 (==、!=、<、<=、>、>=)、逻辑运算符 (&&、||、!) 和 empty 运算符。empty 运算符用于检查对象、字符串和集合是否为空。

可以使用圆括号来组成子表达式。

表达式语言提供了自动类型转换功能和 5 种字面量: 字符串 (用单引号或双引号括起)、整数、浮点数、布尔量 (true 和 false) 以及 null。

在任何表达式中, 都可使用 7 个特殊变量。

- cookie: 由所有 cookies 组成的集合, 其中每个 cookie 都是 javax.servlet.http 包中 Cookie 类的一个实例。

- header: 由所有请求报头组成的集合, 其中每个请求报头都是一个字符串。
- headerValues: 由所有请求报头组成的集合, 其中每个请求报头都是一个字符串数组。
- initParam: 由所有应用程序初始化参数组成的集合, 其中每个参数都是一个字符串。
- pageContext: java.servlet 包中 jspPageContext 类的一个实例。
- param: 由所有请求参数组成的集合, 其中每个参数都是一个字符串。
- paramValues: 由所有请求参数组成的集合, 其中每个参数都是一个字符串数组。

使用上述集合或其他数据结构时, 可以使用操作 c:foreach 来简化遍历其中每个元素的工作。下面的示例显示随 JSP 页面一起发送的所有报头变量:

```
<c:forEach items='${header}' var='head'>
  <ul>
    <li>Name: <c:out value='${head.key}' /></li>
    <li>Value: <c:out value='${head.value}' /></li>
  </ul>
</c:forEach>
```

下面 4 个变量可用于显式地引用特定作用域内的变量。

- applicationScope: 由作用域为整个应用程序的对象组成的集合。
- pageScope: 由作用域为当前页面的对象组成的集合。
- requestScope: 由作用域为当前请求的对象组成的集合。
- sessionScope: 由作用域为当前会话的对象组成的集合。

例如, 表达式 \${sessionScope.price} 检索作用域为当前会话、名为 price 的对象。如果在其他作用域中没有名为 price 的对象, 则也可以使用表达式 \${price}。

本章的最后一个项目将演示 JSTL 的强大功能和灵活性。这个页面使用 XML 操作来显示来自一个 RSS 新闻网站的信息。RSS 是一种 XML 格式, 用于以机器能够识别的格式提供 Web 内容。

JSTL 能够导入指定 URL 处的 XML、HTML 和任何其他数据, 并对其进行分析, 即使该 URL 与使用 JSTL 的页面位于不同的服务器上。

RSS 是 Really Simple Syndication 的缩写, 它让网站能够彼此共享新闻提要、链接和其他内容, 而读者使用一个名为 RSS 集结器 (aggregator) 的软件来阅读网站内容。程序清单 21.13 是网站 SportFilter 使用的 RSS feed 的简化版本。

程序清单 21.13 完整的 sportsfilter.rss 的源代码

```
1: <rss version="2.0">
2:   <channel>
3:     <title>SportsFilter</title>
4:     <link>http://www.sportsfilter.com/</link>
5:     <docs>http://www.rssboard.org/rss-specification</docs>
6:     <item>
7:       <title>Babe Ruth used steroids?</title>
8:       <link>http://sports.espn.go.com/mlb/news/story?id=1745899</link>
9:       <description>Houston Astros second baseman Jeff Kent said the
10: steroids controversy is an embarrassment to baseball and that the
11: public needs to rethink whether sports heroes of yore abstained
12: from illegal performance-enhancing drugs.</description>
13:     </item>
```

```

14: <item>
15:   <title>Phoenix sports fans shell out the bucks</title>
16:   <link>http://washpost.com/wp-dyn/articles/A10394-2007Feb26.html</link>
17:   <description>Phoenix-area taxpayers have invested $700 million
18: in new stadiums for their pro baseball, basketball, football, and
19: hockey franchises, a world record for governmental sports support,
20: as described in today's Washington Post.</description>
21: </item>
22: <item>
23:   <title>Just buy it</title>
24:   <link>http://www.nike.com/nikebiz/news/pressrelease.jhtml</link>
25:   <description>Marion Jones competing in floor gymnastics, Randy
26: Johnson bowling, Lance Armstrong in the boxing ring, Andre Agassi
27: fielding 2B for the Red Sox. Nike rolls out their spring campaign
28: asking in some rather creatively edited commercial spots starting
29: tonight.</description>
30: </item>
31: </channel>
32: </rss>

```

SportFilter 是一个流行的体育社区博客，其网址为 <http://www.sportsfilter.com>，它使用 RSS 2.0 feed 来共享最新的新闻。对于 feed 中的 XML 数据，可使用核心库中的 `c:import` 操作将其读入到一个变量中，并使用 XML 库中的 `x:parse` 操作对其进行分析。

对于分析后的数据，可以对其进行检查、过滤、转换和显示。程序清单 21.14 中 JSP 应用程序使用简单 XPath 语句来提取并显示部分 XML 数据。

程序清单 21.4 rss.jsp 的完整源代码

```

1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2: <html>
3: <head>
4:   <!-- Declare the two tag libraries used on this page -->
5:   <%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>
6:   <%@ taglib uri='http://java.sun.com/jsp/jstl/xml' prefix='x' %>
7:
8:   <!-- Import the RSS feed at the URL specified by feed -->
9:   <!-- For example -->
10:  <!-- http://sportsfilter.com/rss.cfm -->
11:  <c:import url='${param.feed}' var='feedData' />
12:
13:  <!-- Parse the RSS feed -->
14:  <x:parse xml='${feedData}' var='feed' />
15:
16:  <!-- Retrieve the channel element -->
17:  <x:set select='${feed}/rss/channel' var='channel' />
18:
19:  <!-- Display the channel element's title -->
20:  <title><x:out select='${channel}/title' /></title>
21: </head>
22:
23: <body>
24:
25:  <!-- Display the channel element's link and title -->
26:  <p>Headlines from <a href='${channel}/link' /></a>
27:  <x:out select='${channel}/title' /></a>
28:
29:  <!-- Loop through the item elements -->
30:  <p><ul>
31:    <x:forEach select='${feed}/rss/channel/item'>
32:      <!-- Display each element's link, title, description -->
33:      <!-- Descriptions may contain HTML. -->
34:      <li><a href='${channel}/link' /><x:out select='${channel}/link' /></a>
35:      <x:out select='${channel}/description' escapeXml='false' /></li>
36:    </x:forEach>
37:  </ul>
38:
39:  </body>
40: </html>

```

页面中的注释对其中的操作进行了描述。

在 JSTL 中, 最难掌握的是表达式语言, 通过使用采用 JSP 语法的用于 Java 语句和表达式的标记库版本, 可避开表达式语言。

JSTL 的表达式语言同其他 JSP 语法之间的区别在于其功能方面的限制。它没有提供赋值运算符和条件逻辑, 开发人员必须使用外部 Java 类和 JSTL 操作来完成这些任务。

虽然有些程序员不能接受仅仅为了编写 Web 应用程序而学习一种新语言的想法, 但表达式语言很简单, 程序员很快便能掌握它, 对那些熟悉 JavaScript 的人更是如此。

通过避免将 Java 代码放在 JSP 页面中, 表达式语言提供了可重用性和可靠性。它与 JSTL 一道, 使 JSP 离其将 Web 应用程序内容和生成内容的代码分开的目标更近了一步。

JSTL 是 Struts 的补充, 后者是一个开源 Web 应用程序框架, 它也是 Apache Jakarta 项目的一项成果。

#### 警告

信奉 Struts 遵循的模型视图控制器 (model-view-controller) 理念的程序员可能会对 JSTL 包含数据库操作提供质疑, 因为它们并不属于应用程序的表示层。

除非 Web 应用程序非常简单, 否则程序员应考虑将数据库访问行为放在类中, 然后通过 JSP 来使用它们, 而不应使用 JSTL 的 SQL 操作。

## 21.5 总结

至此, 您有 3 种在 Web 上使用 Java 语言的方式: 小程序、servlet 和 JSP。

javax.servlet 和 javax.servlet.http 包中的类主要用于同 Web 服务器交换信息。可使用 Java servlet 来替换 CGI, 后者是最为流行的使用编程语言在 Web 上检索和提供数据的方式。

由于 servlet 可以使用除图形用户界面外的所有 Java 语言特性, 因此您可以使用它们来创建复杂的 Web 应用程序。

JSP 用于将网页中的静态内容同 servlet 生成的动态内容分开。通过使用表达式、语句和声明, 可以在服务器页面上编写 Java 程序, 甚至无需编译程序、将其放到网页中、设计界面或发布类文件。

JSP 标准标记库 (JSTL) 及其表达式语言在将内容和代码分开的道路上再向前迈出了一步, 让 Web 应用程序能够以基于标记的方式使用 Java, 就像使用 HTML 和 XML 一样。

在过去的 3 周中, 读者学习了 Java 语言的语法和 Java 类库中的核心类, 还探索了由其他公司和开发商提供的优秀类库, 如 XOM (XML 对象模型库) 和 JSTL。

现在有能力攻克真正的难题: 使用 Java 类从空白开始编写出健壮、可靠的程序。

祝您编程愉快!

## 21.6 问与答

问: 能够让 Java 小程序同 servlet 进行通信吗?

答: 要让小程序连接到 servlet 后继续运行, servlet 必须与包含小程序的网页位于同一台机器中。出于安全方面的考虑, 小程序不能连接到除托管它的机器之外的其他机器。

要让小程序在 Web 浏览器中加载 servlet, 可调用小程序的 getAppletContext() 方法, 以获得一个 AppletContext 对象, 然后调用该对象的 showDocument(URL) 方法, 并将 servlet 的 URL 作为参数传递给它。

问: 为何 servlet 和 JSP 需要通过 getRealPath() 方法来判断文件在 Web 服务器的什么位置? 能否将文件存储在 servlet 所在的文件夹中并使用不包含路径的文件名吗?

答: Tomcat 不支持在 servlet 或 JSP 中使用相对文件名。要读写文件中的数据, 必须知道该文件在 Web 服务器的什么位置。由于在 Web 托管环境中, 您并非总是知道这种信息, 因此接口 `ServletContext` 包含方法 `getRealPath()`, 该方法向 Web 服务器询问文件的完整路径名。与 CGI 脚本相比, 使用 Tomcat 最大的优点之一是, 您可以直接与服务器进行通信。

在本章的示例 `counter.jsp` 中, `counter.dat` 文件位于 `counter.jsp` 所在的文件夹中。Tomcat 不会将该文件存储到 servlet 所在的文件夹中。

## 21.7 小测验

请回答下述问题, 以复习本章介绍的内容。

### 21.7.1 问题

- 如果 servlet 同时被 5 个 Web 用户运行, 则该 servlet 的 `init()` 方法将被调用多少次?
  - 5;
  - 1;
  - 0~1。
- JSP 页面中的 `request` 变量引用的是 `javax.servlet.http` 中的哪个类?
  - `HttpServletResponse`;
  - `HttpServletRequest`;
  - `servletContext`。
- 哪个 JSP 元素使用标记 `<%=和%>`?
  - 声明;
  - 表达式;
  - 语句。

#### 答案

- c, Web 服务器首次加载 servlet 时, `init()` 方法将被调用。在这 5 个用户请求 servlet 之前, servlet 可能已被加载, 因此 `init()` 可能被调用一次, 也可能一次也不会被调用。
- b,
- b, 这些标记中的表达式将被计算, 结果将显示在表达式所在的位置。

### 21.7.2 认证练习

下面的问题是 Java 认证考试中可能出现的问题, 请回答该问题, 而不要查看本章的内容, 也不要使用 Java 编译器对代码进行测试。

对于下述代码:

```
public class CharCase {
    public static void main(String[] arguments) {
        float x = 9;
        float y = 5;
        char c = '1';
        switch (c) {
            case 1:
                x = x + 2;
            case 2:
                x = x + 3;
```



```
        default:
            x = x + 1;
    }
    System.out.println("Value of x: " + x);
}
```

x 被显示时，其值为多少？

- a. 9.0;
- b. 10.0;
- c. 11.0;
- d. 该程序无法通过编译。

答案 b

## 21.8 练习

为巩固本章介绍的知识，请尝试完成下面的练习。

1. 创建一个 servlet，将表单中的数据保存到一个文件中。
2. 编写一个 JSP 页面，它向 Internet Explorer 用户显示一句问候语，而对其他用户显示另一句问候语。





## 附录

附录 A 使用 Java 开发包

附录 B 使用 Java 开发包编程

附录



## 附录 A

# 使用 Java 开发包

为方便开发，Sun Microsystems 提供了一个免费的 Java 程序开发工具：Java 开发包（Java Development Kit, JDK）。

JDK 是一组用于创建、编译和运行 Java 程序的命令程序。每次发布新的 Java 版本时，Sun 都会提供新的开发包，最新 JDK 版本为 6。

虽然市面上有更高级的 Java 编程工具，如 Borland JBuilder、IntelliJ IDEA 和 Eclipse，但很多程序员仍然使用 JDK。多年来，JDK 一直是笔者使用的主要 Java 编程工具。

本附录介绍如何下载、安装和设置 JDK 以及如何使用它来创建、编译和运行 Java 程序。

另外，还将针对 Java 初学者介绍如何修复最常见的问题——未正确地配置 JDK。

### A.1 选择 Java 开发工具

如果您使用的是 Microsoft Windows 或 Apple Mac OS 系统，其中可能安装了能够运行 Java 程序的 Java 解释器。

要开发 Java 程序，不仅需要解释器，还需要编译器以及用于创建、运行和测试程序的其他工具。

JDK 包含编译器、解释器、调试器、文件归档程序以及几个其他的程序。

JDK 比其他开发工具简单。它没有提供图形用户界面、文本编辑器以及很多程序员需要使用的其他特性。

要使用 JDK，请在文本提示符下执行命令。MS-DOS、Linux 和 UNIX 用户应熟悉这种提示符，它也被称为命令行。

下面是一个命令示例，使用 JDK 时，您可能输入这样的命令：

```
javac RetrieveMail.java
```

它命令程序 javac（JDK 中的 Java 编译器）读取源代码文件 RetrieveMail.java，并创建一个或多个类文件。类文件包含编译后的字节码，可被 Java 解释器执行。

RetrieveMail.java 被编译时，生成的文件之一名为 RetrieveMail.class。如果类文件是一个应用程序，Java 解释器将能够运行它。

熟悉命令行环境的用户在使用 JDK 时将很顺手；其他人在编写程序时，必须习惯没有图形环境的状态。

如果有其他 Java 开发工具，且与 Java 6 兼容，则无需使用 JDK。很多开发工具都可用来创建本书的示例程序。

#### 警告

如果对兼容性有疑问或者这是您首次接触 Java 语言，则可能应该使用 JDK。

另一个免费的开发工具是 NetBeans，可从 Sun 公司网站下载它和 JDK。NetBeans 提供了图形用户界面，它在 JDK 上运行。

## 安装 JDK

JDK 可从 Sun 的 Java 网站下载，网址为 <http://java.sun.com>。

该网站的 Download 区提供了到多个 JDK 版本的链接，还提供了开发环境 NetBeans 及其他与 Java 相关的产品。应下载的产品是 Java Software Development Kit version 6。如果在 Download 区找不到它，可在 Early Access 区查找，这里提供 beta 版。

该 JDK 可用于以下平台：

- Windows 98/Me/NT/2000/XP/Server 2003/Vista;
- Solaris SPARC and Intel;
- Linux。

该 JDK 要求计算机的处理器至少为 Pentium 166MHz，内存为 32MB，可用硬盘空间为 400MB。

### 提示

用于 Macintosh 的 JDK 可直接从 Apple 公司的网站下载。

当您查看该产品时，可能发现 JDK 的版本号由两个数字组成，如 JDK 6.1。为修复 bug 和安全方面的问题，Sun 定期发布新的 JDK 版本，并通过在主版本号加上句点和数字对其进行编号。请选择最新的 JDK 6 版本，不管其编号是 6.0、6.1、6.2 还是更高。

### 警告

不要从 Sun 公司的网站下载两个名称类似的产品：the Java Runtime Environment (JRE) 6.0 和 the Java Standard Edition 6.0 Source Release。

要安装 JDK，必须下载并运行一个安装程序或从光盘安装。在 Sun 的网站，选择针对您的操作系统的 JDK 版本后，可将其作为单个文件进行下载。

下载该文件后，便可以开始安装 JDK 了。

### 在 Windows 平台上安装

安装 JDK 之前，必须确保系统中没有安装其他 Java 开发工具（假设您不需要其他工具）。安装多个 Java 编程工具通常会导致 JDK 出现配置方面的问题。

要在 Windows 系统上安装该程序，可双击安装文件，也可单击“开始”按钮并选择“运行”，再找到并运行该文件。

InstallShield Wizard 将引导您完成软件安装过程。接受 Sun 提出的使用该 JDK 的条款和要求后，需要指定程序的安装位置，如图 A.1 所示。

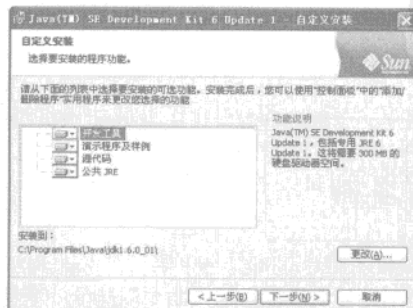


图 A.1 安装 JDK

向导推荐了一个安装文件夹。在图 A.1 中, 向导推荐的文件夹为 C:\Program Files\Java\jdk1.6.0\_01。当您安装 JDK 时, 推荐的文件夹可能与此不同。

要选择其他文件夹, 可以单击“更改”按钮, 然后选择或新建一个文件夹, 再单击“确定”按钮。向导将返回“自定义安装”对话框。

#### 提示

进入下一步之前, 将选择的文件夹名称记录下来。后面配置 JDK 和修复配置问题时需要用到。

安装程序还可能询问您要安装 JDK 的哪些组件。默认情况下, 向导将安装所有的 JDK 组件。

- 开发工具: 可执行程序, 用于创建 Java 程序。
- 演示程序及样例: 可运行的 Java 程序及其源代码, 您可以通过查看这些源代码来学习 Java。
- 源代码: Java 类库中成千上万个类的源代码。
- 公共 JRE: 一个 Java 解释器, 您可将其随程序一起分发。

如果接受默认设置, 将需要大约 350MB 的可用硬盘空间。可以忽略除程序文件外的其他项, 以节省硬盘空间。然而, 演示程序和 Java 运行环境很有用, 因此如果有足够的磁盘空间, 最好也安装它们。

本书不需要源代码, 只有经验丰富的 Java 程序员对它们感兴趣。

要删除不想安装的组件, 可单击它旁边的硬盘图标, 然后选中“现在不安装此功能”选项。

选择要安装的组件后, 单击“下一步”按钮。安装向导将询问是否将 Java 插件安装到 Web 浏览器中。

Java 插件是一个 Java 解释器, 用于运行网页中的 Java 程序。这种程序被称为小程序, 可被不同的解释器运行, 但大多数浏览器没有支持最新 Java 版本的解释器。Sun 提供了一个 Java 插件, 它支持最新的 Java 版本, 可用于 Microsoft Internet Explorer、Mozilla Firefox、Opera 和其他浏览器。

完成配置后, 向导将在您的系统上安装 JDK。

## A.2 配置 JDK

安装 JDK 后, 您必须编辑计算机的环境变量, 以便包含对 JDK 的引用。

要设置 JDK, 经验丰富的 MS-DOS 用户可以调整两个变量, 然后重启系统。

- 编辑计算机的 PATH 变量, 添加到 JDK 的 bin 文件夹 (如果 JDK 安装在文件夹 C:\Program Files\Java\jdk1.6.0 中, 则为 C:\Program Files\Java\jdk1.6.0\bin) 的引用。
- 编辑 CLASSPATH 变量, 使其包含一个到当前文件夹的引用 (句点和分号) 以及指向 JDK 的 lib 文件夹中文件 tools.jar 的引用 (如果 JDK 安装在文件夹 C:\Program Files\Java\jdk1.6.0 中, 则为 C:\Program Files\Java\jdk1.6.0\lib\tools.jar)。

对于不熟悉 MS-DOS 的用户, 可参阅下一节的内容, 该节详细介绍了如何在 Windows 系统上设置变量 PATH 和 CLASSPATH。

对于使用其他操作系统的用户, 应参阅 Sun 公司在 JDK 下载页面提供的指南。

### A.2.1 使用命令行界面

使用 JDK 时, 必须通过命令行来编译和运行 Java 程序及处理其他任务。

命令行提供了通过键盘 (而不是使用鼠标) 输入命令以操纵计算机的方式。当前, Windows 程序很少需要使用命令行。

**注意**

在 Windows 系统中，要切换到命令行，可执行如下操作：

- 对于 Windows 95/98/Me，单击“开始”按钮，然后选择“程序”>“MS-DOS 方式”。
- 对于 Windows NT/2000，单击“开始”按钮，然后选择菜单“程序”>“附件”>“命令提示符”。
- 对于 Windows XP，单击“开始”按钮，然后选择“所有程序”>“附件”>“命令提示符”。

选择上述菜单后，将打开一个命令行窗口，可以在其中输入命令。

Windows 命令行使用 MS-DOS (Windows 之前的 Microsoft 操作系统) 命令。MS-DOS 支持 Windows 的功能：复制、移动和删除文件和文件夹、运行程序、扫描和修复硬盘、格式化软盘等。

图 A.2 是一个命令行窗口。



图 A.2 使用命令行窗口

如果允许输入命令，该窗口的命令行将有一个不断闪烁的光标。在图 A.2 中，命令行为 C:\Document and Setting\Rogers>。

由于在 MS-DOS 窗口中，可以删除文件甚至格式化硬盘，因此尝试使用其命令之前，您应对 MS-DOS 操作系统有基本的了解。

然而，就使用 JDK 而言，您只需了解几个 MS-DOS 操作：如何创建文件夹、如何切换到文件夹、如何运行程序。

### A.2.2 切换文件夹

在 Windows 系统中使用 MS-DOS 时，可以访问在 Windows 中能够使用的所有文件夹。例如，如果 C 盘上有一个 Windows 文件夹，则在 MS-DOS 提示符下，该文件夹为 C:\Windows。

要在 MS-DOS 中切换到某个文件夹，可执行 CD 命令并指定文件夹名，然后按回车键，如下例所示：

```
CD C:\TEMP
```

执行上述命令后，将切换到 C 盘的 TEMP 文件夹（如果有这样的文件夹的话）。切换到某个文件夹后，命令行将为该文件夹的名称，如图 A.3 所示。



图 A.3 在命令行窗口中切换文件夹

也可以通过其他方式使用 CD 命令。

- CD\：切换到当前硬盘驱动器的根目录。
- CD foldername：切换到当前文件夹的子文件夹 foldername（如果有这样的文件夹）。
- CD..：切换到当前文件夹的父文件夹。例如，如果当前文件夹为 C:\Windows\Fonts，则执行命令 CD..后，将切换到文件夹 C:\Windows。

建议您创建一个名为 J21work 的文件夹，用于存储本书的程序。如果已经创建了该文件夹，可以使用下述命令切换到该文件夹：

1. CD\；
2. CD J21work。

如果还没有创建该文件夹，可在 MS-DOS 中完成这项任务。

### A.2.3 在 MS-DOS 中创建文件夹

要从命令行中创建文件夹，可使用 MD 命令和要创建的文件夹名称，然后按回车键，如下例所示：

```
MD C:\STUFF
```

这将在 C 盘的根目录中创建文件夹 STUFF。要切换到刚创建的文件夹，可使用 CD 命令和该文件夹的名称，如图 A.4 所示。



图 A.4 在命令行窗口中创建一个文件夹，然后切换到该文件夹

如果还没有创建文件夹 J21work，可在命令行中完成这项任务：

1. 切换到根目录中（使用 CD\命令）；
2. 输入命令 MD J21work 并按回车键。

创建文件夹 J21work 后，可使用命令 CD\J21work 切换到该文件夹。

为使用 JDK，您需要学习的最后一项 MS-DOS 知识是如何运行程序。

#### A.2.4 在 MS-DOS 中运行程序

要在命令行中运行程序，最简单的方式是输入它的名称并按回车键。例如，输入 DIR 并按回车键，可以查看当前文件夹中的文件和子文件夹。

您也可以在程序名后加上空格和控制程序如何运行的选项。这些选项被称为参数。

来看一个这样的例子。切换到根目录（使用 CD\），然后输入 DIR J21work。您将看到一个列表，其中包含 J21work 目录中的所有文件和子目录。

安装 JDK 后，应运行 Java 解释器，看它能否正常工作。在命令行中输入如下命令：

```
java -version
```

其中，java 是 Java 解释器的名称，-version 是一个参数，它命令解释器显示其版本号。

该命令的输出如图 A.5 所示，但您的版本号可能与此不同，这取决于您安装的是哪个 JDK 版本。



图 A.5 在命令行窗口中运行 Java 解释器

如果 java -version 正常运行，并显示了版本号，则它应以 1.6 打头。版本号可能包含 3 个数字，但只要它以 1.6 打头，便说明使用的 JDK 版本是正确的。

如果运行 java -version 后，显示的版本号不正确或显示错误消息 Bad command or filename，则需要对 JDK 的配置进行修改。

#### 警告

Sun Microsystems 总是让人对其 Java 名称和版本感到迷惑。虽然最新的 Java 版本名为 Java 6，且 JDK 名为 JDK 6.0，但 JDK 的内部版本号为 1.6.0。命令 -version 显示的以及 JDK 的默认安装文件夹都是该内部版本号。

如果其他所有措施都以失败告终，请运行命令 java -version，并核实安装的开发工具是否正确。如果版本号以 1.6 打头，则说明安装的工具是正确的，可用于开发 Java 6 程序。

### A.2.5 修复配置错误

当您首次编写 Java 程序时，错误的原因很可能不是输入错误、语法错误或其他编程错误。大多数错误是由于 JDK 配置不正确引起的。

如果在命令行上执行 `java -version` 命令，而系统找不到包含 `java.exe` 的文件夹，将出现下面的错误消息或与之类似的错误消息（这取决于您使用的操作系统）：

```
Bad command or file name
'java' is not recognized as an internal or external command, opera-
ble program, or batch file
```

要修复这种错误，必须配置系统的 PATH 变量。

### 1. 在 Windows 98/Me 中设置 PATH 变量

在 Windows 98/Me 系统中, 可通过编辑主盘根目录中的 AUTOEXEC.BAT 文件来配置 PATH 变量。MS-DOS 使用该文件来设置环境变量和配置命令行程序如何工作。

AUTOEXEC.BAT 是一个文本文件，可使用 Windows 中的“记事本”编辑它。要启动“记事本”，可单击 Windows 任务栏中的“开始”按钮，然后选择菜单“程序”>“附件”>“记事本”。

启动“记事本”后，选择菜单“文件”>“打开”，然后切换到主盘的根目录，并打开文件AUTOEXEC.BAT。

打开该文件后，将看到一系列 MS-DOS 命令，每个命令占一行，如图 A.6 所示。

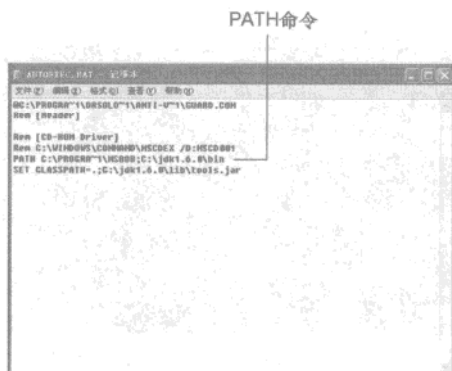


图 A.6 用“记事本”编辑文件 AUTOEXEC.BAT

应查找那些以 PATH 开头的命令。

PATH 命令的后面是一个空格和一系列用分号分隔的文件夹名称。它设置 PATH 变量——包含命令行程序的文件列表。

当您在命令行中执行程序时，PATH 用于帮助 MS-DOS 找到该程序。在前面的例子中，图 A.6 中的 PATH 命令包含两个文件夹：

- C:\PROGRA~1\MSBOB;
- C:\jdk1.6.0\bin。

可在命令行执行如下命令来查看 PATCH 变量的设置：

PATH

要正确设置 JDK，必须在文件 AUTOEXEC.BAT 的 PATH 命令包含 Java 解释器所在的文件夹。解释器的文件名为 java.exe。如果 JDK 1.6 安装在文件夹 C:\Program Files\Java\jdk1.6.0 中，java.exe 将位于文件夹 C:\Program Files\Java\jdk1.6.0\bin 中。

如果记不起 JDK 的安装位置，可查找 java.exe，方法是单击“开始”按钮，然后菜单“查找/文件或文件夹”。可能多个文件夹中都包含这样的文件。要确定哪个是正确的，可在命令行窗口对每个这样的文件执行下述操作：

1. 使用 CD 命令切换到 java.exe 所在的文件夹；
2. 运行命令 java -version。

知道正确的文件夹后，在 AUTOEXEC.BAT 文件的末尾创建一个空行，并在其中添加如下代码：

```
PATH rightfoldername;%PATH%
```

例如，如果正确的文件夹为 c:\java\jdk1.6.0\bin，则在 AUTOEXEC.BAT 的末尾添加下面一行：

```
PATH c:\Program Files\Java\jdk1.6.0\bin;%PATH%
```

%PATH%用于避免覆盖 AUTOEXEC.BAT 中的其他 PATH 命令。在该命令中，使用双引号将文件夹名 Program Files 括起来了，因为有些 Windows 版本要求采用这种方式处理包含空格的文件夹名。

修改完 AUTOEXEC.BAT 后，保存它，然后重启计算机并运行 java -version 命令。

如果显示的 JDK 版本正确，则说明系统的配置可能是正确的。当您在附录后面创建示例程序时，将确定这一点。

## 2. 在 Windows NT/2000/XP/2003 中设置 PATH 变量

在 Windows NT/2000/XP/2003 中，变量 PATH 是通过“环境变量”对话框（“控制面板”的属性之一）来配置的。

要打开该对话框：

1. 在桌面上的“我的电脑”图标上单击鼠标右键或单击“开始”按钮，然后选择“属性”打开“系统属性”对话框；
2. 单击“高级”标签；
3. 单击“环境变量”按钮打开“环境变量”对话框，如图 A.7 所示。

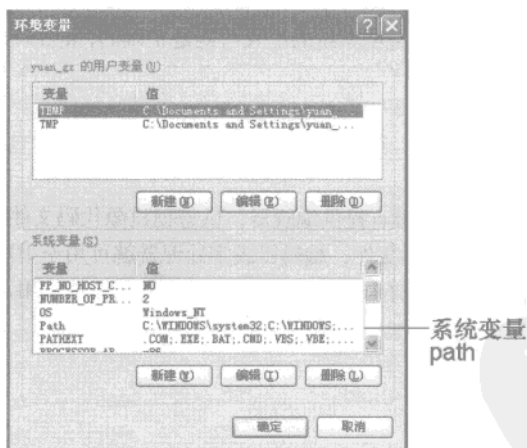


图 A.7 在 Windows NT/2000/XP/2003 中设置环境变量

可编辑的环境变量有两种：系统变量和用户变量。前者用于计算机上的所有用户，后者只用于当前用户。

PATH 是一个系统变量，当您在命令行运行程序时，它帮助 MS-DOS 查找该程序。该变量包含一系列由分号分隔的文件夹。

要正确设置 JDK，必须在文件 AUTOEXEC.BAT 的 PATH 命令包含 Java 解释器所在的文件夹。解释器的文件名为 java.exe。如果 JDK 安装在文件夹 C:\Program Files\Java\jdk1.6.0 中，java.exe 将位于文



件夹 C:\Program Files\Java\jdk1.6.0\bin 中。

如果记不起 JDK 的安装位置,可查找 java.exe,方法是单击“开始”按钮,然后在菜单中选择“搜索”。可能多个文件夹中都包含这样的文件。要确定哪个是正确的,可在命令行窗口对每个这样的文件执行下述操作:

1. 使用 CD 命令切换到 java.exe 所在的文件夹;
2. 运行命令 java -version。

知道正确的文件夹后,返回到“环境变量”对话框,选择“系统变量”列表框中的 Path,然后单击“编辑”按钮。这将打开“编辑系统变量”对话框,其中的“变量名”文本框的内容为 PATH,而文本框“变量值”中是一系列文件夹,如图 A.8 所示。

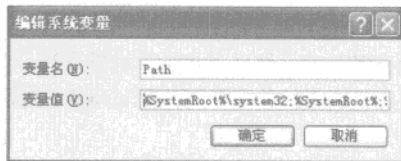


图 A.8 修改 PATH 变量

要将文件夹添加到 PATH 变量中,单击文本框“变量值”,并将光标移到末尾。然后,输入一个分号和 Java 解释器所在的文件夹的名称。

例如,如果正确的文件夹为 C:\Program Files\Java\jdk1.6.0\bin,则在 PATH 变量末尾添加如下文本:

```
c:\Program Files\Java\jdk1.6.0\bin
```

修改后,单击 OK 按钮两次:一次关闭“编辑系统变量”对话框,另一次关闭“系统变量”对话框。

然后,打开命令行窗口并执行命令 java -version。

如果显示的 JDK 版本是正确的,则说明系统的配置可能是正确的,虽然只有到本附录后面使用 JDK 时才能确定这一点。

### A.3 使用文本编辑器

同高级 Java 开发工具不同, JDK 没有提供编辑器,供您创建源代码文件时使用。

只要能够保存没有任何格式的文本文件,编辑器或字处理器就可用于 JDK。

这种特性的名称随编辑器而异。保存文档或设置文档的属性时,请使用下述格式选项之一:

- 纯文本;
- ASCII 文本;
- DOS 文本;
- 文本 (Text-only)。

如果您使用的是 Windows,可使用该操作系统包含的几种编辑器。

“记事本”是一个简单的文本编辑器,只能处理纯文本文件。它只能一次处理一个文档。在 Windows XP 中,要运行“记事本”,可单击“开始”按钮,然后选择“所有程序/附件/记事本”菜单。在其他 Windows 系统中,请单击“开始”按钮,然后选择“程序”>“附件”>“记事本”。

“写字板”要比“记事本”高级些。它能够同时处理多个文档,且能够处理纯文本文件和 Word 文档。它还能够记住最后处理的几个文档,并将它们的名称列在菜单“文件”中。与“记事本”一样,它也位于“附件”中。

Windows 用户还可以使用 Microsoft Word, 但必须将文件保存为文本格式, 而不是 Word 专用格式。UNIX 和 Linux 用户可以使用 emacs、pico 和 vi 来编写程序; 而 Macintosh 可使用 SimpleText 或前面提到的 UNIX 工具编写 Java 源代码文件。

使用诸如“写字板”和“记事本”等简单文本编辑器的缺点之一是, 当您编辑时它们不会显示行号。

在 Java 编程中, 行号很有帮助, 因为很多编译器都通过行号指出错误的位置。请看下述由 JDK 编译器生成的错误消息:

```
Palindrome.java:2: Class Font not found in type declaration.
```

其中 Java 源代码文件名后面的数字 2 指出了导致编译器错误的行。使用支持行号的文本编辑器时, 您可以直接跳到这一行, 并查找错误。

通常, 使用商用 Java 编程软件包来调试程序更佳, 但 JDK 用户必须使用 javac 指出的行号来查找编译错误。这是使用 JDK 学习 Java 后应使用高级 Java 开发工具的重要原因之一。

#### 提示

另一种选择是使用具备行号和其他特性的文本编辑器。jEdit 是最流行的 Java 编辑器之一, 这是一个免费的编辑器, 可用于 Windows、Linux 和其他操作系统。

笔者个人推荐使用 UltraEdit。对程序员和 Web 设计人员来说, 这是一个优秀的编辑器。

## A.4 创建程序

安装并设置 JDK 后, 便可以创建 Java 程序, 以确保 JDK 能正常工作。

Java 程序开始为源代码——用文本编辑器创建的一系列语句, 被保存为文本文件。您可以使用任何程序来创建这种文件, 只要它能够将文件保存为没有任何格式的纯文本。

JDK 没有文本编辑器, 但大多数 Java 开发工具都内置了用于创建源代码文件的编辑器。

请运行您选择的编辑器, 并输入程序清单 A.1 中的 Java 程序代码。请正确地输入其中的圆括号、花括号和引号, 同时确保大小写完全与该程序清单相同。如果您使用的编辑器要求输入文本前提供文件名, 请提供 HelloUser.java。

#### 程序清单 A.1 HelloUser.java 的源代码

```
1: public class HelloUser {
2:     public static void main(String[] arguments) {
3:         String username = System.getProperty("user.name");
4:         System.out.println("Hello " + username);
5:     }
6: }
```

在该程序清单中, 左边的行号和冒号并非程序的组成部分, 这里提供它们旨在方便引用程序中的行。

输入上述程序后, 将其保存到硬盘中, 文件名为 HelloUser.java。保存 Java 源代码文件时, 其扩展名必须是 .java。

#### 提示

如果您创建了文件夹 J2lwork, 请将 HelloUser.java 和本书中其他所有的 Java 源代码文件都保存到该文件夹中。这样, 使用命令行窗口时, 查找它们将更容易。

如果您使用的是 Windows, 则诸如“记事本”等文本编辑器可能在您保存的 Java 源代码文件名后加上扩展名.txt。例如, HelloUser.java 被保存为 HelloUser.java.txt。为避免这种问题, 可在保存源代码文件时, 将文件名用引号括起。图 A.9 说明了如何在“记事本”中使用这种技术将文件保存为 HelloUser.java。

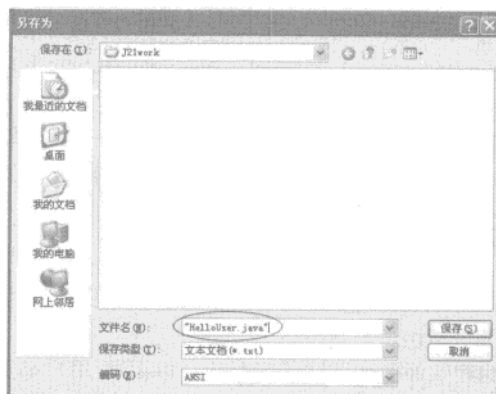


图 A.9 在“记事本”中保存源代码文件

**提示**

一种更佳解决方案是，将 .java 文件与使用的文本编辑器关联起来。在 Windows 中，打开 HelloUser.java 所在的文件夹，然后在该文件上双击。如果您没有打开过扩展名为 .java 的文件，则系统将询问您要使用哪个程序来打开这样的文件。选择您喜欢的编辑器，并选中使这种选择永久化的选项。这样，以后要打开源代码文件进行编辑时，只需双击它即可。

这个程序旨在测试 JDK，本附录不会介绍这个 6 行的程序使用的任何 Java 编程概念。在本书前几章中，您将学习有关 Java 的基本知识。

### A.4.1 在 Windows 中编译和运行程序

现在，可以使用 JDK 中的 Java 编译器（名为 javac 的程序）对源代码进行编译。编译器读取扩展名为 .java 的源代码文件，并创建一个或多个可被 Java 解释器运行的 .class 文件。

请打开命令行窗口，切换到 HelloUser.java 所在的文件夹。

如果该文件位于主盘根目录的 J21work 文件夹中，则可以使用下述命令来切换到该文件夹：

```
cd \J21work
```

切换到正确的文件夹中后，可以在命令提示符下输入下述命令来编译 HelloUser.java：

```
javac HelloUser.java
```

图 A.10 显示了用于切换到文件夹 J21work 并编译 HelloUser.java 的 MS-DOS 命令。

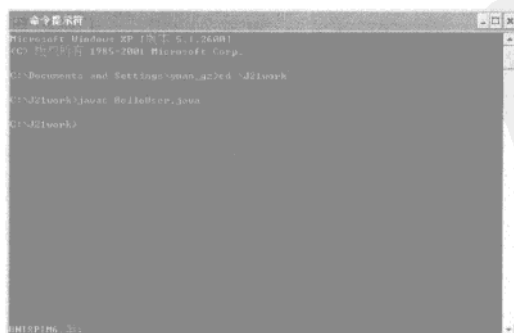


图 A.10 在命令行窗口中编译 Java 程序

如果程序通过编译，JDK 编译器将不会显示任何消息；如果程序有错误，编译器将指出错误以及导致错误的代码行。

如果该程序编译时没有发生任何错误，将创建一个名为 `HelloUser.class` 的文件，它位于 `HelloUser.java` 所在的文件夹中。

类文件中包含将被 Java 解释器执行的 Java 字节码。如果发生错误，请查看源代码文件，确保您输入的内容与程序清单 A.1 完全相同。

创建类文件后，可以使用 Java 解释器来运行它。JDK 中的解释器名为 `java`，它也是从命令行运行的。

要运行程序 `HelloUser`，请切换到 `HelloUser.class` 所在的文件夹，然后执行下述命令：

```
java HelloUser
```

您将看到 `Hello`、逗号和您的用户名。

#### 注意

使用 JDK 的 Java 解释器运行 Java 类时，请不要在类名后指定文件扩展名 `.class`。如果这样做，将看到与下面类似的错误消息：

```
Exception in thread "main" java.lang.
  NoClassDefFoundError: HelloUser/class
```

图 A.11 显示了用于编译、运行该程序的命令以及该程序的输出。

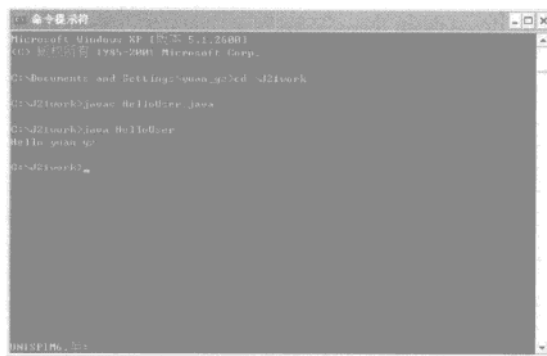


图 A.11 运行 Java 应用程序

如果能够成功地编译和运行该程序，说明 JDK 能正常工作，您可以开始阅读本书第 1 章。

如果即使程序中的代码与书中完全相同，程序仍无法通过编译，则问题可能是 JDK 的配置不正确，您可能需要配置环境变量 `CLASSPATH`。

### A.4.2 设置 `CLASSPATH` 变量

所有的 Java 程序都依赖于两种类：您创建的类和 Java 类库，后者是数百个表示 Java 语言功能的类。

JDK 需要知道到哪里去查找 Java 类文件。很多情况下，JDK 在其安装目录中查找来解决这种问题。您可以创建或修改环境变量 `CLASSPATH` 来设置查找位置。

#### 1. 在 Windows 98/Me 中设置 `CLASSPATH`

如果成功地编译和运行了程序 `HelloUser`，说明 JDK 已被正确配置，无需对系统做任何修改。

另一方面，如果运行程序时，看到错误消息 `Class not found error` 或 `NoClassDefFound`，则必须确保

变量 CLASSPATH 的设置是正确的。

为此，可运行“记事本”，选择菜单“文件”>“打开”，然后切换到系统的根目录，并打开文件 AUTOEXEC.BAT。这样，编辑器将打开一个包含多个 MS-DOS 命令的文件，如图 A.12 所示。

请找到该文件中包含命令 SET CLASSPATH= 的一行。该命令的后面是一系列用分号分隔的文件夹和文件名。

CLASSPATH 用于帮助 Java 编译器查找其所需的类文件。图 A.12 中的 SET CLASSPATH= 命令包含两项，这两项由分号隔开：

```
c:\jdk1.6.0\lib\tools.jar
```

CLASSPATH 中可以包含文件和文件夹，还可以包含句点——MS-DOS 中另一种引用当前文件夹的方式。

要查看系统的 CLASSPATH 变量，可在命令行执行下述命令：

```
ECHO %CLASSPATH%
```



图 A.12 编辑系统的环境变量

如果 CLASSPATH 中包含不存在的文件或文件夹，应将其引用从 AUTOEXEC.BAT 的 SET CLASSPATH= 行中删除，同时务必删除多余的分号。

要正确地设置 JDK，必须在 SET CLASSPATH= 命令中包含 Java 类库所属的文件。该文件名为 tools.jar。如果 JDK 安装在文件夹 C:\Program Files\Java\jdk1.6.0 中，则 tools.jar 可能位于文件夹 C:\Program Files\Java\jdk1.6.0\lib 中。

如果不记得 JDK 的安装位置，可以单击 Windows 任务栏中的“开始”按钮，然后选择“查找/文件或文件夹”菜单，以查找 tools.jar。如果有多个这样的文件，可以使用下述方法来判断哪个是正确的：

1. 使用 CD 命令切换到 Java 解释器 (java.exe) 所在的文件夹；
2. 执行命令 CD..；
3. 执行命令 lib。

通常，正确的 tools.jar 位于该 lib 文件夹中。

知道正确的位置后，在文件 AUTOEXEC.BAT 的末尾创建一个空行，并添加以下内容：

```
SET CLASSPATH=%CLASSPATH%;.;rightlocation
```

例如，如果文件 tools.jar 位于文件夹 C:\Program Files\Java\jdk1.6.0\lib 中，则应在文件 AUTOEXEC.BAT 的末尾添加下面的一行：

```
SET CLASSPATH=%CLASSPATH%;.;c:\Program Files\Java\jdk1.6.0\lib\tools.jar
```

修改好 AUTOEXEC.BAT 后，将其保存并重新启动计算机。然后在此编译并运行程序 HelloUser。正确设置 CLASSPATH 变量后，您应该能够完成这项任务。

## 2. 在 Windows NT/2000/XP/2003 中设置 CLASSPATH

在 Windows NT/2000/XP 系统中，也可以使用“环境变量”对话框来配置 CLASSPATH 变量。

要打开该对话框：

1. 在桌面上的“我的电脑”图标上单击鼠标右键或单击“开始”按钮，然后选择“属性”打开“系统属性”对话框；
2. 单击“高级”标签；
3. 单击“环境变量”按钮打开“环境变量”对话框，如图 A.13 所示。

如果您系统中包含变量 CLASSPATH，则它可能是系统变量之一。您的系统中可能没有变量 CLASSPATH，JDK 通常不需要该变量就能找到类文件。

然而，如果系统中有 CLASSPATH，则它必须至少包含两项内容：到当前文件夹的引用（句点）以及到 Java 类库所属文件（tools.jar）的引用。

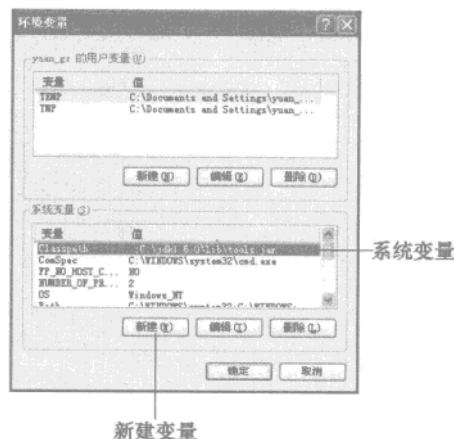


图 A.13 在 Windows NT/2000/XP/2003 中设置环境变量

如果 JDK 安装在文件夹 C:\Program Files\Java\jdk6.0 中，则 tools.jar 将位于文件夹 C:\Program Files\Java\jdk6.0\lib 中。

如果不记得 JDK 的安装位置，可以单击 Windows 任务栏中的“开始”按钮，然后选择“搜索”菜单来查找 tools.jar。如果有多个这样的文件，可以使用下述方法来判断哪个是正确的：

1. 使用 CD 命令切换到 Java 解释器（java.exe）所在的文件夹；
2. 执行命令 CD..；
3. 执行命令 lib。

通常，正确的 tools.jar 位于该 lib 文件夹中。

知道正确的文件夹后，返回到图 A.13 所示的“环境变量”对话框中。

如果系统中没有 CLASSPATH 变量，则单击“系统变量”列表下面的“新建”按钮打开“新建系统变量”对话框。

如果系统中已经有一个 CLASSPATH 变量，则选中它，然后单击“编辑”按钮打开“编辑系统变量”对话框。

这两个对话框包含的内容都相同：一个“变量名”文本框和一个“变量值”文本框。

在“变量名”文本框中输入 CLASSPATH，并在“变量值”文本框中输入 CLASSPATH 的正确值。

例如，如果 JDK 安装在文件夹 c:\Program Files\Java\jdk1.6.0 中，则 CLASSPATH 应为：

```
.;C:\Program Files\Java\jdk1.6.0\lib\tools.jar
```

图 A.14 说明了作者是如何设置自己的系统的。在该系统中, JDK 安装在文件夹 C:\jdk1.6.0 中。

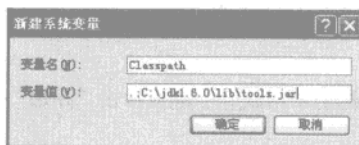


图 A.14 在 Windows XP 中设置 CLASSPATH

设置好 CLASSPATH 后, 单击 OK 按钮两次: 一次关闭“编辑系统变量”或“新建系统变量”对话框, 另一次关闭“环境变量”对话框。

与 Windows 98/Me 不同, 无需重新启动系统, 所做的修改就能生效。请打开命令行窗口, 并执行命令 `java -version`。

如果显示的 JDK 版本正确, 则说明系统被正确配置, 无需做其他的调整。请再次运行程序 `HelloUser`, CLASSPATH 变量被正确配置后, 该程序应该能够正常运行。



## 附录 B

# 使用 Java 开发包编程

本书使用 JDK (Java Development Kit, Java 开发包) 创建、编译和运行 Java 程序。

JDK 包含很多程序员根本不会使用的功能,而您可能对其中的一些工具本身也不熟悉。

本附录介绍 JDK 的一些功能,通过它们您可以创建更可靠、测试更充分、运行速度更快的 Java 程序。

将介绍的主题如下:

- 使用解释器运行 Java 应用程序;
- 使用编译器编译程序;
- 使用 appletviewer 运行 Java 小程序;
- 使用文档工具创建文档;
- 使用调试器查找程序中的 bug 及深入了解其性能;
- 使用解释器和 appletviewer 设置系统属性。

### B.1 JDK 概览

虽然可用于创建 Java 程序的开发环境有几十个,但使用最广泛的是 Sun Microsystems 的 Java 开发包 (JDK),这是一组命令行工具,可用于使用 Java 语言来开发软件。

JDK 流行的两个主要原因是:

- 它是免费的,可从 Sun 公司的官方 Java 网站免费下载,其网址为 <http://java.sun.com>;
- 它最早面世, Sun 发布新的 Java 版本时,首先支持该版本的工具位于 JDK 中。

JDK 使用命令行——在 Windows 中称为 MS-DOS 命令提示符或控制台;在 UNIX 中,称为外壳提示符。命令是使用键盘输入的,如下例所示:

```
javac VideoBook.java
```

上述命令使用 JDK 编译器编译名为 VideoBook.java 的 Java 程序。该命令由两个元素组成:JDK 编译器的名称 (javac) 和要编译的程序的名称 (VideoBook.java),它们之间用空格隔开。

所有 JDK 命令都采用相同的格式:要使用的工具的名称和一个或多个指出工具应做什么的元素。这些元素称为参数。

下面的例子演示了如何使用命令行参数:

```
java VideoBook add VHS "Invasion of the Bee Girls"
```

它命令 Java 解释器运行类文件 VideoBook,并给它提供 3 个命令行参数:字符串 add、VHS 和 Invasion of the Bee Girls。

#### 注意

您可能认为这里不止 3 个命令行参数,因为字符串单词 Invasion of the Bee Girls 中有空格。“Invasion of the Bee Girls”两边的引号使它被视为一个命令行参数,且使得参数可以包含空格。



一些 JDK 参数可限制工具的运行方式。这些参数前面有连字符，被称为选项。

下面的命令演示了如何使用选项：

```
java -version
```

它命令 Java 解释器显示其版本号，而不是运行类文件。这可以用于确定 JDK 是否被正确配置以运行 Java 程序。下面是在安装了 JDK 6 的系统上执行该命令得到的输出：

```
java version "1.6.0"
Java(TM) 2 Runtime Environment, Standard Edition (build1.6.0)
Java HotSpot(TM) Client VM (build 1.6.0, mixed mode)
```

这里显示的版本是在 Sun 公司内部的 JDK 6 版本号——1.6。

有时候，可以结合使用选项和其他参数。例如，编译使用了被摒弃的方法的 Java 类时，可以使用选项 `-deprecation`，以查看关于这些方法的更详细的信息，如下所示：

```
javac -deprecation OldVideoBook.java
```

## B.2 java 解释器

java (Java 解释器) 用于从命令行运行 Java 应用程序。它接受一个参数——要运行的类文件名，如下例所示：

```
java BidMonitor
```

虽然 Java 类文件的扩展名为 `.class`，但使用解释器时不应指出扩展名。

Java 解释器运行的类必须包含一个 `main()` 方法，该方法的格式如下：

```
public static void main(String[] arguments) {
    // Method here
}
```

有些简单 Java 程序只有一个类：包含 `main()` 方法的类。对于使用了其他类的复杂程序，解释器将自动加载所需的其他类。

Java 解释器运行的是字节码——由 Java 虚拟机执行的编译后的指令。当 Java 程序被作为 `.class` 文件保存为字节码后，可被不同的解释器执行，而无需做任何修改。编译后的 Java 程序与任何支持 Java 的解释器都兼容。

### 注意

有意思的是，Java 并非可用于创建 Java 字节码的唯一一种语言。NetRexx、JPython、JRuby、JudoScript 以及其他几十种语言都可使用专用的编译器编译为可执行字节码的 `.class` 文件。Robert Tolksdorf 维护了一个这些语言的列表，可从网页 <http://www.robet-tolksdorf.de/vmlanguages> 找到。

要指定由 Java 解释器运行的类文件，有两种方式。如果类不位于任何包中，可以通过指定其名称来运行它，如前面的 `java BuyItem` 示例所示；如果类位于某个包中，则必须指定完整包名和类名。

例如，假设 `SellItem` 类位于 `org.cadenhead.auction` 包中。要运行该应用程序，需要使用下面的命令：

```
java org.cadenhead.auction.SellItem
```

包名的每个部分对应于相应的子文件夹。Java 解释器将在下面几个地方查找 `SellItem.class` 文件：

- 当前文件夹中的 `org\cadenhead\auction` 子文件夹（如果当前文件夹为 `C:\J21work`，且 `SellItem.class` 位于文件夹 `C:\J21work\org\cadenhead\auction` 中，则它将被成功执行）；
- Classpath 设置中任何文件夹的 `org\cadenhead\auction` 子文件夹。

如果要创建自己的包，则一种简单的管理方式是，将一个文件夹加入到 Classpath 中，该文件夹是您创建的所有包的根目录，如 `C:\javapackages`。创建对应于包名的子文件夹后，将这个包的类文件放到正确的子文件夹中。

Java 支持断言——一种仅在用户通过命令行启用时才会发挥作用的功能。要使用 Java 解释器运行

程序并利用该程序中的断言，可使用命令行参数-`ea`，如下例所示：

```
java -ea Outline
```

Java 解释器将执行应用程序类及其使用的其他类文件中的所有 `assert` 语句，但 Java 类库中的类除外。

要取消这种例外，以利用所有的断言，可在运行类文件时使用选项-`esa`。

如果没有指定启用断言功能的选项，解释器将忽略所有的 `assert` 语句。

## B.3 编译器 javac

Java 编译器 `javac` 将 Java 源代码转换为一个或多个由字节码组成的类文件，这些类文件可被 Java 解释器执行。

Java 源代码存储在扩展名为 `.java` 的文件中。这种文件可使用任何能够将文档保存为纯文本的文本编辑器或字处理程序来创建。这里使用的术语随文本编辑软件而异，但这种文件通常被称为纯文本、ASCII 文本、DOS 文本等。

Java 源代码文件可包含多个类，但只有一个可被声明为公有的。如果您需要，Java 源代码文件甚至可以不包含任何公有类，虽然由于继承规则，小程序不可能这样。

如果源代码文件包含被声明为公有的类，则该文件名必须与该类名相同。例如，公有类 `BuyItem` 的源代码必须存储在文件 `BuyItem.java` 中。

要编译文件，可运行 `javac` 工具，并使用该文件的名称作参数，如下所示：

```
javac BuyItem.java
```

可以编译多个源文件，方法是把这些文件的名称作为命令行参数并用空格将其隔开，如下面的命令所示：

```
javac BuyItem.java SellItem.java
```

还可以使用通配符（如 `*` 和 `?`）。下面的命令编译某个文件夹中的所有 `.java` 文件：

```
javac *.java
```

编译一个或多个 Java 源代码文件时，对于每个被成功编译的 Java 类，都将创建一个 `.class` 文件。

如果被编译的程序使用了断言，则必须使用选项-`source 1.6`，如下面的命令所示：

```
javac -source 1.6 Outline.java
```

如果编译包含断言的程序时没有使用选项-`source`，`javac` 将显示一条错误消息，且不对文件进行编译。

运行编译器时，另一个很有用的选项是-`deprecation`，它使编译器对 Java 程序中使用的已被摒弃的方法进行描述。

被摒弃的方法指的是 Sun Microsystems 已经用更好的方法替代它，这个方法位于同一个类中或其他类中。虽然被摒弃的方法仍然可用，但 Sun 公司在某个时候可能决定将其从类中删除。摒弃警告强烈建议您尽早停止使用这种方法。

通常情况下，如果程序使用了已摒弃的方法，编译器将发出警告。-`deprecation` 选项导致编译器列出每个已被摒弃的方法，如下面的命令所示：

```
javac -deprecation SellItem.java
```

如果您更在乎 Java 程序的运行速度，而不是其类文件的大小，可以使用选项-`O` 来编译其源码。这将创建一个被优化以提高执行速度的类文件。静态的、`final` 或私有的方法可能被编译为内联的 (`inline`)，内联技术增加了类文件的体积，但使得方法的执行速度更快。

如果要使用调试器来查找 Java 类中的 bug，可在编译源代码时使用选项-`g`，以便将所有的调试信息（包括行号引用、局部变量和源代码）加入到类文件中；要防止这些内容被加入到类文件中，可在编译时使用选项-`g:none`。

通常情况下, Java 编译器在创建类文件时, 不会提供大量的信息。实际上, 如果源代码被成功编译, 且没有使用任何已摒弃的方法, 编译器将不会有任何输出。在这里, 没有消息就是好消息。

要获悉 javac 工具在编译源代码时执行的操作的更详细信息, 可使用 -verbose 选项, 这样将描述用于完成各种功能的时间、加载的类以及总共所需的时间。

## B.4 浏览器 appletviewer

appletviewer 用于运行需要 Web 浏览器且是超文本标记语言 (HTML) 文档的一部分的 Java 程序, 它接受 HTML 文档作为命令行参数, 如下例所示:

```
appletviewer NewAuctions.html
```

如果参数是 Web 地址, 而不是对文件的引用, appletviewer 将加载该地址处的 HTML 文档。例如:

```
appletviewer http://www.javaonthebrain.com
```

图 B.1 是一个从该网站加载的小程序, 该网站是漫画家兼 Java 游戏开发人员 Karl Homell 开发的。

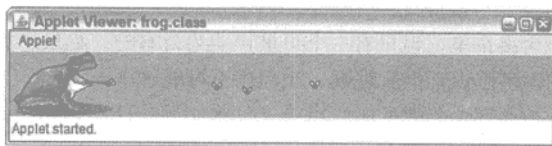


图 B.1 使用 appletviewer 运行 Java 小程序

HTML 文档被 appletviewer 加载后, 其中的每个小程序都将在自己的窗口中运行: 这些窗口的大小取决于相应小程序的 HTML 标记中的 HEIGHT 和 WIDTH 属性的值。

与 Web 浏览器不同, appletviewer 不能用于查看 HTML 文档本身。要查看小程序相对于文档中的其他内容的位置, 必须使用支持 Java 的 Web 浏览器。

### 警告

最新的 Mozilla Firefox 和 Microsoft Internet Explorer 本身不支持 Java 小程序, 但可使用 Sun Microsystems 的浏览器插件来支持 Java 小程序。Sun 的 Java Plug-in 可替代浏览器的 Java 解释器来运行 Java 小程序。Java Plug-in 包含在 Java Runtime Environment 中, 这是一个随 JDK 一起安装 Java 程序解释器。如果您的系统没有安装 Java Plug-in, 可从 Sun 的网站 <http://java.sun.com> 下载。

appletviewer 的用法非常简单, 但您可能对查看器运行小程序时可用的某些菜单项不熟悉。

下面的菜单项都是可用的。

- Restart 和 Reload 用于重新启动小程序。这两个选项之间的差别在于, Restart 在重新启动小程序之前不会卸载它, 而 Reload 会这样做。Reload 选项相当于首先关闭小程序, 然后在网页中再次打开它。

- Start 和 Stop 选项用于直接调用小程序的 start() 和 stop() 方法。
- Clone 选项创建小程序的第二个拷贝, 它运行在自己的窗口中。
- Tag 选项显示程序的 applet 或 object 标记以及配置小程序的 param 标记的 HTML 代码。

Applet 菜单中的另一个选项是 Info, 它调用小程序的 getAppletInfo() 和 getParameterInfo() 方法。程序员可以实现这些方法, 以提供有关小程序及其处理的参数的详细信息。方法 getAppletInfo() 返回一个描述小程序的字符串; 方法 getParameterInfo() 返回一个字符串数组的数组, 它指出了每个参数的名称、类型和描述。

程序清单 B.1 的 Java 小程序演示了如何使用这些方法。

**程序清单 B.1 完整的 AppInfo.java 源代码**

```

1: import java.awt.*;
2:
3: public class AppInfo extends javax.swing.JApplet {
4:     String name, date;
5:     int version;
6:
7:     public String getAppletInfo() {
8:         String response = "This applet demonstrates the "
9:             + "use of the Applet's Info feature.";
10:        return response;
11:    }
12:
13:    public String[][] getParameterInfo() {
14:        String[] p1 = { "Name", "String", "Programmer's name" };
15:        String[] p2 = { "Date", "String", "Today's date" };
16:        String[] p3 = { "Version", "int", "Version number" };
17:        String[][] response = { p1, p2, p3 };
18:        return response;
19:    }
20:
21:    public void init() {
22:        name = getParameter("Name");
23:        date = getParameter("Date");
24:        String versText = getParameter("Version");
25:        if (versText != null) {
26:            version = Integer.parseInt(versText);
27:        }
28:    }
29:
30:    public void paint(Graphics screen) {
31:        Graphics2D screen2D = (Graphics2D) screen;
32:        screen2D.drawString("Name: " + name, 5, 50);
33:        screen2D.drawString("Date: " + date, 5, 100);
34:        screen2D.drawString("Version: " + version, 5, 150);
35:    }
36: }

```

该小程序的主要功能是显示 3 个参数的值：Name、Date 和 Version。方法 `getAppletInfo()` 返回如下字符串：

This applet demonstrates the use of the Applet's Info feature.

如果您没有使用过多维数组，则 `getParameterInfo()` 方法要复杂些。它执行如下操作：

- 第 13 行将方法的返回类型定义为二维的 String 对象数组。
- 第 14 行创建一个包含 3 个元素的 String 对象数组，这 3 个元素分别是 Name、String 和 Programmer's Name。这些元素描述了小程序 AppInfo 的一个参数。它们描述了参数的名称 (Name)、参数的数据类型 (字符串) 以及参数说明 (Programmer's Name)。这个包含 3 个元素的数组存储在对象 p1 中。

- 第 15~16 行定义了另外两个 String 数组，用于描述参数 Date 和 Version。
- 第 17 行使用对象 response 来保存一个包含 3 个字符串数组 (p1、p2 和 p3) 的数组。
- 第 18 行将对象 response 用作方法的返回值。

程序清单 B.2 的网页可用于加载小程序 AppInfo。

**程序清单 B.2 完整的 AppInfo.html 源文件**

```

1: <applet code="AppInfo.class" height="200" width="170">
2: <param name="Name" value="Rogers Cadenhead">
3: <param name="Date" value="12/07/06">
4: <param name="Version" value="5">
5: </applet>

```

图 B.2 显示了使用 appletviewer 运行该小程序的情况，图 B.3 是选择菜单项 Info 打开的对话框。

要使用这些功能，浏览器必须能够将这些信息提供给用户。JDK 中的 appletviewer 通过菜单项 Info 来处理这项工作，但诸如 Internet Explorer 等浏览器当前并不提供这样的功能。

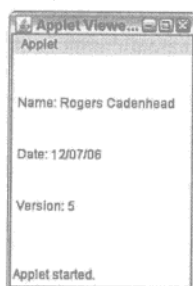


图 B.2 运行在 appletviewer 中的小程序 AppInfo

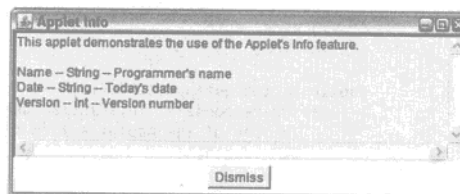


图 B.3 小程序 AppInfo 的 Info 对话框

## B.5 文档工具 javadoc

javadoc 是 Java 的文档生成器，它以 java 源代码文件或包名为输入，生成一个 HTML 格式的详细文档。

要使 javadoc 能够为程序创建完整的文档，必须在程序的源代码中使用一种特定的注释语句。本书的程序使用 `//`、`/*` 和 `*/` 来创建注释——帮助人们理解程序的信息。

Java 还有一种结构化程度更高的注释，可被 javadoc 读取。这种注释用于描述程序中的元素，如类、变量、对象和方法，其格式如下：

```
/** A descriptive sentence or paragraph.
 * @tag1 Description of this tag.
 * @tag2 Description of this tag.
 */
```

Java 文档注释应放在它所说明的程序元素前面，并简洁地说明该程序元素。例如，如果注释位于 class 语句之前，它应描述这个类的用途。

除描述性文本外，还可以使用其他条目来进一步说明程序元素。这些条目被称为标记 (tag)，它们的前面是 `@`，后面是一个空格和描述性句子或段落。

程序清单 B.3 是小程序 AppInfo 的详细说明版本，名为 AppInfo2。该程序使用了如下标记：

- **@author**：程序的作者。该标记只能用于类，如果运行 javadoc 时没有使用 `-author` 选项，这种标记将被忽略。
- **@version text**：程序的版本号。该标记也只能用于类，如果运行 javadoc 时没有使用 `-version` 选项，这种标记将被忽略。
- **@return text**：对方法返回的变量和对象进行说明。
- **@serial text**：描述变量或可被序列化（将对象及其变量的值存储到磁盘中，供以后检索）的对象的数据类型和可能值。

### 程序清单 B.3 完整的 AppInfo2.java 源代码

```
1: import java.awt.*;
2:
3: /** This class displays the values of three parameters:
4:  * Name, Date and Version.
5:  * @author <a href="http://java21days.com/">Rogers Cadenhead</a>
6:  * @version 5.0
7:  */
8: public class AppInfo2 extends javax.swing.JApplet {
9:     /**
10:      * @serial The programmer's name.
11:      */
12:     String name;
13:     /**
14:      * @serial The current date.
```

```

15:    */
16:    String date;
17:    /**
18:     * @serial The program's version number.
19:     */
20:    int version;
21:
22:    /**
23:     * This method describes the applet for any browsing tool that
24:     * requests information from the program.
25:     * @return A String describing the applet.
26:     */
27:    public String getAppletInfo() {
28:        String response = "This applet demonstrates the "
29:            + "use of the Applet's Info feature.";
30:        return response;
31:    }
32:
33:    /**
34:     * This method describes the parameters that the applet can take
35:     * for any browsing tool that requests this information.
36:     * @return An array of String[] objects for each parameter.
37:     */
38:    public String[][] getParameterInfo() {
39:        String[] p1 = { "Name", "String", "Programmer's name" };
40:        String[] p2 = { "Date", "String", "Today's date" };
41:        String[] p3 = { "Version", "int", "Version number" };
42:        String[][] response = { p1, p2, p3 };
43:        return response;
44:    }
45:
46:    /**
47:     * This method is called when the applet is first initialized.
48:     */
49:    public void init() {
50:        name = getParameter("Name");
51:        date = getParameter("Date");
52:        String versText = getParameter("Version");
53:        if (versText != null) {
54:            version = Integer.parseInt(versText);
55:        }
56:    }
57:
58:    /**
59:     * This method is called when the applet's display window is
60:     * being repainted.
61:     */
62:    public void paint(Graphics screen) {
63:        Graphics2D screen2D = (Graphics2D)screen;
64:        screen.drawString("Name: " + name, 5, 50);
65:        screen.drawString("Date: " + date, 5, 100);
66:        screen.drawString("Version: " + version, 5, 150);
67:    }
68: }

```

可通过下面的命令使用源代码文件 AppInfo2.java 来创建 HTML 文档：

```
javadoc -author -version AppInfo2.java
```

Java 文档工具将创建多个网页，这些网页存储在 AppInfo2.java 所在的文件夹中。这些页面以 Sun 的 Java 类库文档的方式对该程序进行说明。

#### 提示

要查看 Java 6 和 Java 类库的官方文档，可访问 <http://java.sun.com/javase/6/docs/api/>。

要查看 javadoc 为 AppInfo2 创建的文档，可在 Web 浏览器中加载新创建的网页 index.html。图 B.4 显示了在 Mozilla Firefox 中加载该页面的结果。

javadoc 生成的网页包含大量的超链接。通过这些网页，可以知道文档注释和标记中的信息在哪里。

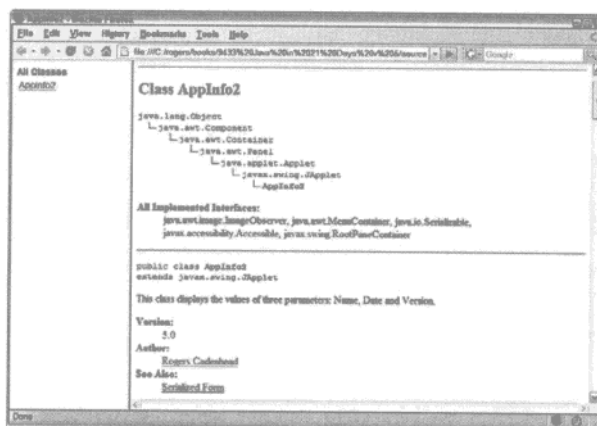


图 B.4 程序 AppInfo2 的 Java 文档

如果您熟悉 HTML，可在文档注释中使用诸如 A、TT 和 B 等标记。

javadoc 还可用于说明整个包，方法是包名作为命令行参数。这将为包中的每个.java 文件创建 HTML 文档，同时创建一个对这些页面进行索引的 HTML 文件。

如果希望生成的 Java 文档被存储到非默认文件夹中，可使用-d 选项，并在后面加上空格和文件夹名称。

下面的命令为 AppInfo2 创建 Java 文档，并将这些文档存储到文件夹 C:\JavaDocs\中：

```
javadoc -author -version -d C:\JavaDocs\ AppInfo2.java
```

下面详细地列出了可在 Java 文档注释中使用的其他标记。

- @deprecated text：指出类、方法或变量已被摒弃。这将导致使用该特性的程序被编译时，javadoc 编译器将发出摒弃警告。
- @exception class description：用于可能引发异常的方法，指出异常的类名及其描述。
- @param name description：用于方法，指出参数的名称及其可能的取值。
- @see class：指出另一个类的名称，将被转换为一个超链接——指向这个类的 Java 文档。该标记在注释中的使用不受任何限制。
- @see class#method：指出另一个类的方法名，将被转换为一个超链接——指向该方法的 Java 文档。其使用也不受任何限制。
- @since text：描述方法或特性是何时被加入到类库中的。

## B.6 Java 文件存档工具 jar

部署 Java 程序时，跟踪程序所需的所有类文件和其他文件是极其繁琐的。

为简化这项工作，JDK 提供了一个名为 jar 的工具，它将程序的所有文件打包成一个 Java 存档文件——也叫 JAR 文件。jar 工具也可用于将这种存档文件进行拆包。

打包成 JAR 文件时，可以使用 Zip 格式进行压缩，也可以不进行任何压缩。

要使用该工具，请输入命令 jar 以及命令行选项和一系列的文件名、文件夹名或通配符。

下面的命令将一个文件夹中的所有类文件和 GIF 图像打包成一个名为 Animate.jar 的 Java 存档文件：

```
jar cf Animate.jar *.class *.gif
```

参数 `cf` 指定了运行 `jar` 程序时可以使用的两个命令行选项。`c` 指出应创建一个 Java 存档文件，而 `f` 指出文件名由接下来的参数指定。

您也可以使用下面这样的命令，将特定的文件加入到 Java 存档文件中：

```
jar cf MusicLoop.jar MusicLoop.class muskratLove.mp3 shopAround.mp3
```

这将创建一个名为 `MusicLoop.jar` 的存档文件，它包含 3 个文件：`MusicLoop.class`、`muskratLove.mp3` 和 `shopAround.mp3`。

运行 `jar` 时，如果没有提供任何参数，将显示可用于该工具的选项列表。

`jar` 的用途之一是，将运行 Java 小程序所需的所有文件都放到一个 JAR 文件中。这样在 Web 上部署该小程序将更容易。

将 Java 小程序加入到网页中的标准方式是，使用标记 `applet` 或 `object` 指出小程序的主类文件。支持 Java 的浏览器将下载并运行该小程序。小程序需要的其他类和文件将从 Web 服务器下载。

以这样的方式运行小程序存在的问题是，对于小程序需要的每个文件——助手类、图像、音频文件、文本文件等，都需要在 Web 浏览器和包含该文件的服务器之间建立一条连接。这可能使下载小程序及其运行所需的文件的时间大大增长。

如果能够将很多文件打包成一个 Java 存档文件，以减少浏览器需要下载的文件数，则 Web 服务器下载和运行小程序的速度将更快。如果 Java 存档文件被压缩，下载速度将更快。

创建 Java 存档文件后，可在 `applet` 标记中使用 `archive` 属性指出存档文件的位置。可通过下面这样的标记来在小程序中使用 Java 存档文件：

```
<applet code="MusicLoop.class" archive="MusicLoop.jar" width="45" height="42">
</applet>
```

该标记指出，小程序使用的文件包含在存档文件 `MusicLoop.jar` 中。支持 JAR 文件的浏览器和浏览工具将在该存档文件中查找小程序运行时所需的文件。

#### 警告

虽然 Java 存档文件中可以包含类文件，但使用 `archive` 属性后，`code` 属性仍然是必不可少的。要运行小程序，浏览器仍然必须知道其主类文件。

使用 `object` 标记来显示使用 JAR 文件的小程序时，小程序的存档文件是通过 `param` 标记中的一个参数来指定的。该标记的 `name` 属性和 `value` 属性应分别为 `archive` 和存档文件的名称。

下面的代码重写了前一个示例，它使用的是 `object`，而不是 `applet`：

```
<object code="MusicLoop.class" width="45" height="42">
  <param name="archive" value="MusicLoop.jar">
</object>
```

## B.7 调试器 jdb

Java 调试器 `jdb` 是一个复杂的工具，用于帮助您查找和修复 Java 程序中的 bug。您还可以使用它来更深入地了解程序运行时在 Java 解释器中发生的情况。它包含大量功能，其中的一些超出了对 Java 语言比较陌生的 Java 程序员的技术范畴。

您不一定非得使用该调试器来调试 Java 程序。这是显而易见的，尤其是当您阅读本书期间创建 Java 程序时。Java 编译器生成错误后，最常见的措施是将源代码加载到编辑器中，找到错误消息指出的代码行，并确定问题所在。这种烦人的编译-报错-查找-修正循环将不断进行下去，直到程序通过编译。

使用这种调试方法一段时间后，您可能认为，对编程而言调试器是多余的，因为这种工具很复杂，不好掌握。修复会导致编译器出错的问题时，这种推理是有道理的。很多这样的问题都是简单错误，



如错放了分号、{和}不匹配或将错误的数据类型作为方法的参数。然而，当您开始查找逻辑错误时——更微妙的 bug，不会导致程序无法编译和运行——调试器将成为宝贵的工具。

Java 调试器的两项功能对于查找其他方式无法发现的 bug 很有用：单步执行和断点。单步执行 (single execution) 在每行代码执行后都暂停；断点 (breakpoint) 是程序暂停的位置。使用 Java 调试器时，断点由指定的代码行、方法调用或捕获的异常触发。

Java 调试器的工作原理是，使用它能够完全控制的 Java 解释器来运行程序。

使用 Java 调试器对程序进行调试之前，应使用 -g 选项来编译该程序，这样，类文件中将包含额外的信息。这些信息对调试的帮助非常大。同样，不应使用 -O 选项，因为优化技术可能生成不与程序源代码直接对应的类文件。

### B.7.1 调试应用程序

要调试应用程序，可运行工具 jdb，并将 Java 类作为参数，如下所示：

```
jdb WriteBytes
```

它使用调试器来调试 WriteBytes.class。

应用程序 WriteBytes 将一系列的字节写入到硬盘中，以创建文件 pic.gif。

调试器将加载该程序，但并不立刻运行它，而是显示如下输出：

```
Initializing jdb...
```

```
>
```

要控制调试器，需要在提示符>下输入命令。

要在程序中设置断点，可使用命令 stop in 和 stop at。命令 stop in 在类中指定方法的第一行设置一个断点。您通过参数来指定类和方法名，如下例所示：

```
stop in SellItem.SetPrice
```

该命令在方法 SetPrice 的第一行设置一个断点。注意，方法名后不需要参数或括号。

命令 stop at 在类中指定行设置一个断点。您通过参数来指定类和行号，如下例所示：

```
stop at WriteBytes:14
```

如果对类 WriteBytes 执行上述命令，您将看到如下输出：

```
Deferring breakpoint WriteBytes:14
```

```
It will be set after the class is loaded.
```

可在类中设置任意数目的断点。要查看当前设置的断点，可使用不带任何参数的 clear 命令。

clear 命令按行号而不是方法名列出当前所有的断点，即使这些断点是使用 stop in 命令设置的。

使用带有类名和行号的 clear 命令可以删除断点。如果方法 SellItem.SetPrice 位于 SellItem 的第 215 行，则可以使用下面的命令来删除这个断点：

```
clear SellItem:215
```

在调试器中，可以使用命令 run 来启动程序。下面是当您开始运行 WriteBytes 类时，调试器的输出：

```
run WriteBytes
```

```
VM Started: Set deferred breakpoint WriteBytes:14
```

```
Breakpoint hit: "thread=main", WriteBytes.main(), line=14 bci=413
```

```
14         for (int i = 0; i < data.length; i++)
```

到达 WriteBytes 类中的断点时，请尝试执行如下命令：

- list：显示断点处的代码行及其前后的几行代码。这需要访问断点所在类的 java 文件，因此 WriteBytes.java 文件必须存储在当前文件夹或 Classpath 列出的文件夹中。

- locals：显示当前使用的或将要定义的局部变量的值。
- print text：显示 text 指定的变量、对象或数组元素的值。
- step：执行下一行，然后停止。
- cont：从断点处开始继续执行程序。

- `!!`: 重复前一个调试命令。

在该应用程序中尝试使用这些命令后，可以删除断点，并使用 `cont` 命令来继续执行该程序。要结束调试，可使用 `exit` 命令。

应用程序 `WriteBytes` 创建一个名为 `pic.gif` 的文件。可以使用 Web 浏览器或图像编辑软件来加载它，以验证该应用程序已成功运行。您将看到一个黑白相间的小型字母 J。

调试完程序，并确保它能正确运行后，别忘了重新编译它，且不要使用选项 `-g`。

## B.7.2 调试小程序

您无法使用工具 `jdb` 加载小程序来调试它，但可以使用 `appletviewer` 的 `-debug` 选项，如下例所示：

```
appletviewer -debug AppInfo.html
```

这将加载 Java 调试器，当您使用诸如 `run` 等命令时，`appletviewer` 将开始运行。请尝试这个例子，以了解这些工具是如何进行交互的。

使用 `run` 命令来执行小程序之前，在方法 `getAppletInfo` 的第一行设置一个断点。这需要使用如下命令：

```
stop in AppInfo.getAppletInfo
```

开始运行该小程序后，仅当您导致方法 `getAppletInfo()` 被调用时，才能到达这个断点。为此，可以选择 `appletviewer` 的菜单项 `Applet/Info`。

## B.7.3 高级调试命令

通过前面介绍的特性，可使用调试器来中止程序的执行和更详细地了解所发生的情况。对很多调试任务而言，这些足够了，但调试器还提供了很多其他的命令。

- `up`: 上移栈帧，以便能够使用 `locals` 和 `print` 命令来查看当前方法被调用之前的情况。
- `down`: 下移栈帧，以便查看方法调用后的情况。

在 Java 程序中，可能调用一连串的方法——一个方法调用另一个方法，而后者又调用另一个方法，以此类推。在每个调用方法的地方，Java 都将作用域中的所有对象和变量组织在一起，以便进行跟踪。这称为堆栈，对象就像一副扑克牌一样堆积在一起。程序运行过程中出现的各种堆栈被称为栈帧 (stack frame)。

结合使用 `locals` 命令和 `up`、`down` 命令，可以更深入地了解调用方法的代码是如何与该方法进行交互的。您还可以在调试过程中使用如下命令。

- `classes`: 列出当前被加载到内存中的类。
- `methods`: 列出类的方法。
- `memory`: 列出内存总量和当前未被使用内存量。
- `threads`: 列出当前执行的线程。

命令 `threads` 将所有进程编号，让您能够使用 `suspend` 命令和进程号来暂停进程，如 `suspend 1`。您可以使用 `resume` 命令和进程号来继续运行指定的进程。

另一种在 Java 程序中设置断点的方式是，使用 `catch text` 命令。捕获 `text` 指定的 `Exception` 类时，程序将暂停执行。

还可以使用 `ignore text` 命令来忽略异常，要忽略的异常由 `text` 指定。

## B.8 使用系统属性

一项晦涩的 JDK 功能是，命令行选项-D 可以调整 Java 类库的性能。

如果学习 Java 之前，您使用过其他编程语言，您可能熟悉环境变量——提供了运行程序的机器使用的操作系统的信息。例如，Classpath 设置指出 Java 解释器应到哪些文件夹中去查找类文件。

由于环境变量的名称随操作系统而异，因此 Java 程序不能直接读取它们。Java 包含大量的、支持 Java 的任何平台都有的系统属性。

由于属性仅用于获取信息。下面的系统属性在任何 Java 实现中都可用。

- java.version: Java 解释器的版本号。
- java.vendor: 一个字符串，指出 Java 解释器的厂商。
- os.name: 当前使用的操作系统。
- os.version: 操作系统的版本号。

用于 Java 程序中时，其他属性将影响 Java 类库的行为。一个这样的例子是属性 java.io.tmpdir，它指定了 Java 输入和输出类将用作临时工作区的文件夹。

属性可通过命令行进行设置，方法是使用-D 选项、属性名、等号(=)和属性的新值，如下面的命令所示：

```
java -Duser.timezone=Asia/Jakarta Auctioneer
```

上面的例子在运行 Auctioneer 类之前，将默认时区设置为 Asia/Jakarta。这将影响 Java 程序中所有没有设置时区的 Date 对象。

这种修改属性的效果不是永久性的，而只影响特定类及其使用的类的执行。

### 提示

在 java.util 包中，TimeZone 类中包含一个名为 getProperties() 的类方法，它返回一个字符串数组，其中包含 Java 支持的所有时区标识符。

下面的语句显示这些标识符：

```
String[] ids = java.util.TimeZone.  
    getAvailableIDs();  
for (int i = 0; i < ids.length; i++) {  
    System.out.println(ids[i]);  
}
```

您还可以创建自己的属性，并使用类 System 的方法 getProperty() 来读取它们，这个类位于 java.lang 包中。

程序清单 B.4 包含一个简单程序的源代码，该程序显示一个用户创建的属性的值。

#### 程序清单 B.4 完整的 ItemProp.java 源代码

```
1: class ItemProp {  
2:     public static void main(String[] arguments) {  
3:         String n = System.getProperty("item.name");  
4:         System.out.println("The item is named " + n);  
5:     }  
6: }
```

运行该程序时，如果没有在命令行上设置 item.name 属性，则输出将如下：

```
The item is named null
```

可使用-D 选项来设置 item.name 属性，如下面的命令所示：

```
java -Ditem.name="Microsoft Bob" ItemProp
```

这样，程序的输出将如下：

```
The item is named Microsoft Bob
```

-D 选项用于 Java 解释器。要将其用于 appletviewer，只需在 -D 前面加上 -J 即可，如下面的命令所示：

```
appletviewer -J-Dtimezone=Asia/Jakarta AuctionSite.html
```

这使得 appletviewer 对网页 AuctionSite.html 上的所有小程序都使用默认时区 Asia/Jakarta。

